UNIVERSIDAD VERACRUZANA

FACULTAD DE INGENIERÍA.

REGIÓN VERACRUZ

"Desarrollo de un Software Libre para la Lectura, Graficación y Registro en Tiempo Real de Señales Provenientes de Ondas Cerebrales"

T E S I S

PARA ACREDITAR LA EXPERIENCIA RECEPCIONAL

Para obtener el Título de:

INGENIERO EN INFORMÁTICA

P R E S E N T A:

Carlos Antonio Bulnes Domínguez

ASESOR DE TESIS

D. Luis Felipe Marín Urías

Boca del Río, Ver. Junio 2015

Índice general

Capítulo 1

Introducción

En la actualidad vivimos en un mundo donde la tecnología es parte esencial de nuestras vidas; La podemos ver alrededor, en nuestros teléfonos celulares, computadoras, relojes, etc. La tecnología está pasando a ser más allá de juguetes personales, en el campo de la ciencia son cada vez más las investigaciones que se realizan para crear tecnología para facilitar la vida cotidiana. Un ejemplo de esto son los sistemas "Interfaz Cerebro-Máquina", BCI por sus siglas en inglés. Las investigaciones con estos sistemas comenzaron en 1970 para la Agencia de Proyectos e Investigación Avanzados de Defensa (DARPA). En la actualidad los sistemas BCI son utilizados principalmente en el área de la medicina principalmente con el fin de ayudar a pacientes con parálisis a comunicarse ya sea restaurando sus movimientos de brazos o piernas o creando sistemas que escriban lo que el paciente esté pensando.
La función de un BCI es la de leer las señales provenientes de la actividad cerebral y enviarla hacía algún dispositivo. Para que este dispositivo sea capaz de recibir e interpretar las señales que recibe del BCI es necesario que se creen aplicaciones en la plataforma que se desea usar (Windows, Linux, OSX, Android, iOS, etc).
Los BCI comerciales vienen equipados con el software necesario para el correcto funcionamiento del BCI, sin embargo, en algunos casos este software resulta muy limitado para los distintos propósitos de investigación además de contar con costos de licencia. En internet empiezan a aparecer algunos proyectos de software libre para remplazar o complementar a los privativos. Uno de los más destacados y que se aborda en este trabajo de tesis es el desarrollado por Cody Brocious y Kyle Machulis llamado Emokit. El Emokit, como se profundizará más a delante, es un software libre desarrollado para leer las señales provenientes del BCI Emotiv Epoc (del cual también se habla). Existe también otro software de nombre "Emotiv Visualizer" basado en Emokit pero añadiendo la característica de graficación de las señales. El Visualizer procesa de manera secuencial y bajo el mismo programa la lectura y graficación de las señales, esto ocasiona que exista inconsistencia en los datos presentados.
En este trabajo de tesis se desarrolla una interfaz basada en el Emokit. Se incorporan funciones y características que facilitar para facilitar el manejo de la información de ondas cerebrales.

1.1. Justificación

La necesidad de la creación de un software libre para la lectura de las ondas cerebrales surge debido a la escaza variedad de programas dedicados a dicha tarea actualmente, donde es el mismo creador del dispositivo físico el que te proporciona el software, el cual, ya cuenta con funciones y operaciones definidas por el fabricante.

En la actualidad los proyectos de investigación requieren interactuar más con la información que obtienen con los dispositivos lectores de ondas cerebrales, y como ya se mencionó, las alternativas actuales se encuentran limitadas, se propone entonces un proyecto de código libre en el cual, partiendo del software desarrollado en este trabajo, los desarrolladores futuros sean capaces de complementarlo e implementarlos a sus necesidades específicas.

1.2. Objetivos

1.2.1. Objetivo General

Crear un software libre capaz de leer y graficar en tiempo real a través de la plataforma ROS las señales enviadas por el Emotiv EPOC, las cuales son interpretadas por un software libre llamado Emokit[1].

1.2.2. Objetivos Particulares

- Establecer un canal de comunicación entre el software y el EEG.
- Obtener datos numéricos a partir de las señales obtenidas.
- Tomar esos datos en tiempo real y graficarlos en una relación frecuencia-tiempo.
- Establecer un canal de salida para enviar los datos interpretados.

1.3. Antecedentes

El software desarrollado en este trabajo toma como base al Emokit[1] el cuál forma parte de OpenYou[1]. El Emokit lee, descifra e interpreta la información enviada por el Emotiv EPOC, el cual es un dispositivo "Interfaz Cerebro-Máquina" (BCI), tales como su nivel de batería , intensidad de la señal, y las 14 lecturas realizadas por el dispositivo; este software actualmente solo imprime a nivel terminal dichos datos.

1.3.1. BCI

Una "Interfaz Cerebro-Máquina"[2] (BCI) es un medio de comunicación directo entre el cerebro y un dispositivo externo. Los BCI están normalmente dirigidos a asistir, aumentar y reparar la habilidad cognitiva y las funciones sensoriomotoras. A pesar del interés existente en la posibilidad de controlar los dispositivos que utilizan directamente las señales del cerebro, ha sido sólo durante los últimos 20 años donde este campo de investigación se ha desarrollado mediante experimentos y publicaciones de interés al respecto, a pesar de que las bases de esta exploración se asentaron a principios del siglo XX[3].
Las investigaciones con los BCI comenzaron en 1970 en la Universidad de Los Ángeles California(UCLA) bajo el subsidio de Fundación Nacional de Ciencia contratados por la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA). La publicación hecha después de la investigación marcó la primera aparición de la expresión "Interfaz Cerebro-Máquina" en la literatura científica.

1.3.1.1. Componentes Esenciales de un BCI

Un BCI está integrada generalmente por: Adquisición de señales, preprocesamiento, extracción de características, clasificación, una interfaz de la aplicación y retroalimentación del sistema.

Adquisición de la Señal
Las señales cerebrales se obtienen mediante un Electroencefalograma[1] (EEG), dado que varias de sus características y formas de onda se relacionan con procesos cerebrales bien conocidos que pueden ser generados espontáneamente por el usuario o inducidos mediante estímulos visuales o sonoros. Como se explicará más adelantes existen diferentes clasificaciones de EEG.

Procesamiento
Una vez obtenidas las señales del EEG estas tienen que ser procesadas mediante diversos pasos:

- Un filtro espacial L se aplica a los datos de dominio de tiempo, representados por una transformación lineal de los datos originales.

[1] Un Electroencefalograma es un análisis que realiza un seguimiento de las ondas cerebrales y las registra.

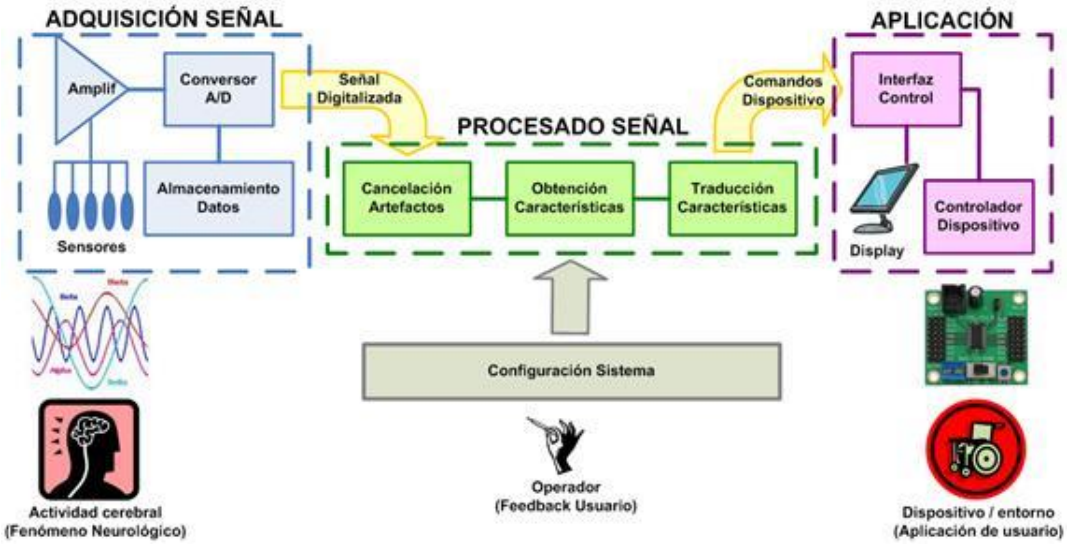

Figura 1.1: Estados básicos al manejar BCI. Primero el BCI adquiere la señal, luego esta es procesada para finalmente llegar a una aplicación con un propósito determinado. Tomado de [3]

- La transformada rápida de Fourier se aplica a los datos filtrados espacialmente. Los datos del EEG se transforman del dominio del tiempo al dominio de frecuencia.
- Finalmente, se seleccionan las bandas de frecuencia más relevantes de acuerdo a la aplicación futura.

El mayor obstáculo para la construcción de BCI potentes basadas en el EEG es la baja relación señal; es decir, que los componentes del EEG que proporcionan información sobre la intención del usuario son por lo general muy encubiertos por la actividad de fondo del cerebro, lo que dificulta en gran medida el análisis de las señales necesarias para una correcta interfaz cerebral.

Aplicaciones
El objetivo de un BCI consiste principalmente en convertir pensamientos en acciones. Las aplicaciones potenciales incluyen interfaces de ordenador, robots móviles, la estimulación muscular funcional, entre otras muchas aplicaciones.

1.3.1.2. Clasificación de los BCI

Existen diferentes formas de clasificar a los BCI.

Por el uso de técnicas invasivas

- BCI invasivos.
- BCI no invasivos.

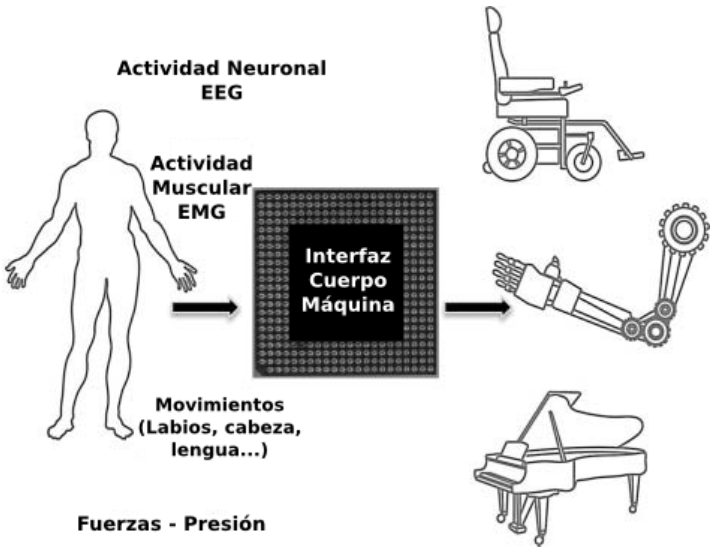

Figura 1.2: Esquemática de una interfaz típica Cerebro-Máquina. Señales obtenidas mediante EEG, Electromiograma (EEM) y medición de fuerzas en las extremidades. Parcialmente tomado de [3]

Por el uso de estimulación

- Endógenos
- Exógenos

Por las características EEG

- Sistemas BCI basados en potenciales corticales lentos
- Sistemas BCI basados en ritmos de actividad cerebral
- Sistemas BCI basados en potenciales evocados

1.3.1.3. BCI Invasivos

Los dispositivos BCI invasivos son implantados directamente en el cerebro y tienen la más alta calidad de señales de los BCI. Estos dispositivos son usados para recuperar funciones de personas con parálisis. Los BCI invasivos son usados también para recuperar la vista conectando el cerebro a cámaras externas y para restaurar el uso de extremidades usando brazos y piernas robóticos controlados por el cerebro. Como todos los dispositivos que se encuentran instalados en la materia gris del cerebro este tipo de BCI produce una calidad de señales muy alta pero son propensos a causar cicatrices en el tejido cerebral causando que las señales comiencen a volverse débiles o incluso la pérdida de las reacciones del cuerpo por contener un objeto desconocido en el cerebro[4].

En las ciencias de la visión los implantes en el cerebro han sido usados para tratar la ceguera no congénita[2]. William Dobell es uno de los primeros científicos que vienen trabajando con una interfaz cerebro para restaurar la vista como una investigación privada[4]. Él implantó su primer prototipo en Jerry, un hombre que quedó ciego en su adultez en 1978. Dobell insertó un BCI de 68 electrodos en la corteza visual de Jerry y logró producir la sensación de ver una luz. En 2012 el experimento fue realizado en Jens Neumann donde Dobell utilizó un implante más sofisticado que permitió un mejor mapeo. Investigadores de la Universidad de Emory en Atlanta, dirigidos por Philip Kennedy y Roy Bakay fueron los primeros en instalar un implante en el cerebro de un ser humano que produce señales de alta calidad suficientes para estimular el movimiento.
Thomas Navin Lal et al. en su artículo [5] desarrollaron un BCI llamado "Dispositivo de Traducción del Pensamiento" o por su nombre en inglés, Thought Translation Device (TTD), el cual usan para ayudar a comunicarse a pacientes con parálisis quienes han perdido sus funciones cognitivas. Para poder usar el TTD los pacientes debieron aprender a regular a voluntad su Slow Cortical Potentials (SCP)[3]. El sistema entonces permite a su usuario escribir textos en la pantalla de una computadora o navegar en internet. El sistema sin embargo cuenta con dos desventajas: no todos los pacientes logran controlar su SCP además la intensidad de la señal es un poco baja y a un usuario bien entrenado requiere aproximadamente 30 segundos para escribir un caracter.
Niels Birbaumer publica en su artículo [7] el resultado de su investigación donde analiza a los BCI invasivos y no invasivos desde la perspectiva de su utilidad clínica para la comunicación y reparación de habilidades motoras en personas con parálisis. En su investigación encontró publicaciones de distintos casos de Esclerosis Lateral Amiotrófica (ELA)[4] en diferentes estados. A los pacientes se les implanto un microelectrodo de vidrio cortical lleno de un factor de crecimiento neurotrófico. Con dicho implante los pacientes lograron comunicarse apagando y encendiendo un flash-ion para decir "si" y "no" sin embargo el autor citado concluye que es complicado realizar un juicio a cerca de los BCI invasivos debido a las complicaciones médicas que puede conllevar. Niels muestra en su publicación 3 tipos de BCI:

- Slow Cortical Potentials: Los pacientes seleccionan una letra de las mostradas en la parte baja de la pantalla, la letra seleccionada aparece en la parte de arriba de la pantalla. Figura 1.3
- SMR BCI: Oscilaciones SMR de la corteza sensoriomotora durante la inhibición del movimiento y las imágenes o ejecución de movimiento (EEG rastro abajo). Figura 1.4

[2]Una enfermedad congénita es aquella que se manifiesta desde el nacimiento, ya sea por un trastorno ocurrido durante el desarrollo embrionario, durante el parto o como consecuencia de un defecto hereditario

[3]Se llaman SCP (o potenciales corticales lentos en español) a los cambios relacionados con los eventos de corriente continua lenta obtenidas con los EEG provenientes de los grandes conjuntos de células en la capa cortical superior[6].

[4]ELA es una enfermedad degenerativa de tipo neuromuscular. Se origina cuando unas células del sistema nervioso llamadas motoneuronas disminuyen gradualmente su funcionamiento y mueren, provocando una parálisis muscular progresiva de pronóstico mortal

- P300[5] BCI: Filas y columnas de letras son resaltadas en sucesiones rápidas, siempre que la letra pensada se encuentre en la cadena iluminada un P300 aparece en el EEG. Figura 1.5

En esta misma publicación se menciona una conferencia sobre BCI realizada en Rennselearville, Nueva York en 2005 donde a los más de 100 científicos que asistieron se les preguntó su opinión sobre el futuro de los BCI. La mayoría opinó que los BCI no invasivos sería el desarrollo más prometedor de la siguiente década.

1.3.1.4. BCI no Invasivos

Los implantes no invasivos son colocados en el cuero cabelludo. Los BCI no invasivos producen señales débiles porque el cráneo amortigua las señales, dispersando las ondas electromagnéticas creadas por las neuronas.
Miguel Ángel López en su tesis doctoral [8], dentro de la clasificación de los BCI describe a los no Invasivos como aquellos métodos que no suponen riesgo alguno para el paciente. Entre los BCI no invasivos se encuentran los EEG, MEG, fMRI y NIRS destacando a los EEG como la alternativa más importante, sencilla y económica de la actualidad. La EEG se basa en el principio de que las corrientes de naturaleza iónica, presentes en la corteza cerebral, producto de la actividad poblaciones de neuronas pueden ser registradas para su análisis mediante el uso de electrodos superficiales extracraneales. Existen también distintas técnicas para la fijación de los electrodos. Una técnica consiste en el uso de un gorro con una serie de agujeros donde ajustar mediante rosca los electrodos (figura 1.6A), esta técnica tiene el inconveniente de tener que ajustar uno a uno cada electrodo, junto con un gel electrolito. Una versión mejorada, Electrocap (figura 1.6B), dispone de los electrodos ya integrados en el gorro y de un sistema cómodo de aplicación del gel. Existen también redes de sensores, como el SensorNet de EGI (figura 1.6C). Este último consiste en una red elástica que se ajusta a la cabeza de cada sujeto con un número muy elevado de sensores (hasta 250).

Javier Danilo et al. describen es su trabajo de titulación [9] el diseño e implementación de un prototipo de BCI para manipular una pinza robótica. A través de electrodos colocados en el cuero cabelludo de una persona y utilizando los principios de la electroencefalografía (EEG) obtienen señales eléctricas pequeñas las cuales someten a etapas de amplificación y filtrado de ruido. Para la adquisición de las señales utilizan la plataforma Arduino, y haciendo uso de la comunicación serial con el software MATLAB para la visualización en la computadora de las señales obtenidas. Posteriormente se identificarán los pulsos eléctricos que son generados por un estímulo visual y que serán los que determinen la activación de un mecanismo. Este mecanismo manipulará a un dispositivo a distancia que en este caso será una pinza robótica (figura 1.7).

El uso de redes o sensores resultan útiles para aplicaciones médicas sin embargo sus interfaces resultan poco atractivas en el ámbito del entretenimiento. Empresas como NeuroSky y Emotiv han desarrollado cascos para el sector del entretenimiento que ofrecen un sistema

[5]La onda P300 es un potencial evocado que puede ser registrado mediante electroencefalografía como una deflexión positiva de voltaje con una latencia de unos 300ms en el EEG

Figura 1.3: Pacientes usando Slow Cortical Potentials

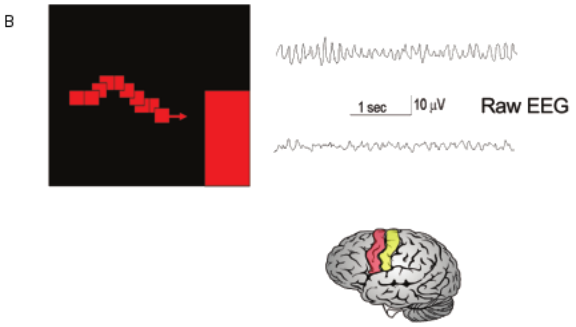

Figura 1.4: BCI de Oscilaciones SMR

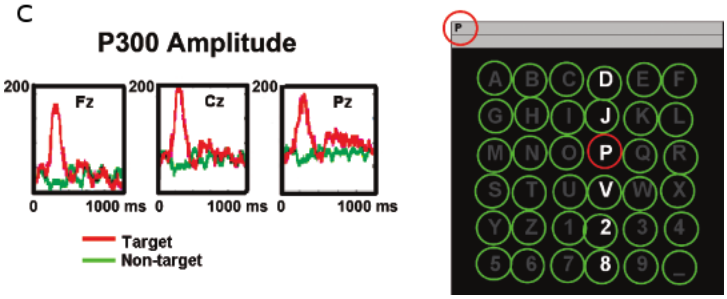

Figura 1.5: BCI basados en la onda P300

Figura 1.6: Sistemas de sujeción de electrodos EEG. A. Electrodos usados en un gorro EEG con agujeros y rosca. B. Electrocap. C. SensorNet de EGI. Tomadas de [8]

rápido, cómodo y portátil. Como se puede observar en la figura 1.8 tanto el NeuroSky como el Emotiv presentan diseños más refinados y prácticos que los BCI usados tradicionalmente. Esto permite que cualquier persona sea capaz de adquirir uno ya sea para fines de entretenimiento, investigación, etc.

Job Ramón de la O Chávez en su tesis "Interfaz Cerebro - Computadora para el Control de un Cursor Basado en Ondas"[10] plantea una interfaz que permita la comunicación entre el usuario y la computadora, haciendo uso de sus ondas cerebrales, para el control de un cursor en pantalla mediante comandos obtenidos de las lecturas de un amplificador de ondas cerebrales. En su trabajo, Chávez describe que la caracterización de las señales se puede hacer mediante 3 componentes de esta, que entregan información diferente tipo pero referente a una misma intención del individuo. Los componentes de la señal que plantean usar son: beta, mu y P300, de las cúales las dos primeras están relacionadas con el movimiento del cuerpo y la última está relacionada con el nivel de concentración del individuo con respecto a lo que está viendo. Chávez propone hacer una combinación lineal de estas tres componentes para obtener información más precisa acerca de lo que el individuo pretende hacer.

1.3.2. Aplicaciones de los BCI

La mayoría de las aplicaciones desarrolladas a la publicación de esta tesis se centran en la restauración de las habilidades cognitivas y motoras de las personas. Existe un sector menor que comienza a utilizar los BCI para el ámbito del entretenimiento y otras áreas. En [8] se presenta un resumen de las distintas aplicaciones para los BCI:

1.3.2.1. Comunicaciones

Esta es la aplicación más importante para personas con discapacidades motoras que se encuentran paralizados teniendo restringidas su comunicación verbal y no verbal. Los pacientes que son aptos para recibir este tipo de tratamiento son aquellos que se encuentran en un estado consiente pero son incapaces de utilizar sus músculos lo que les impide comunicarse

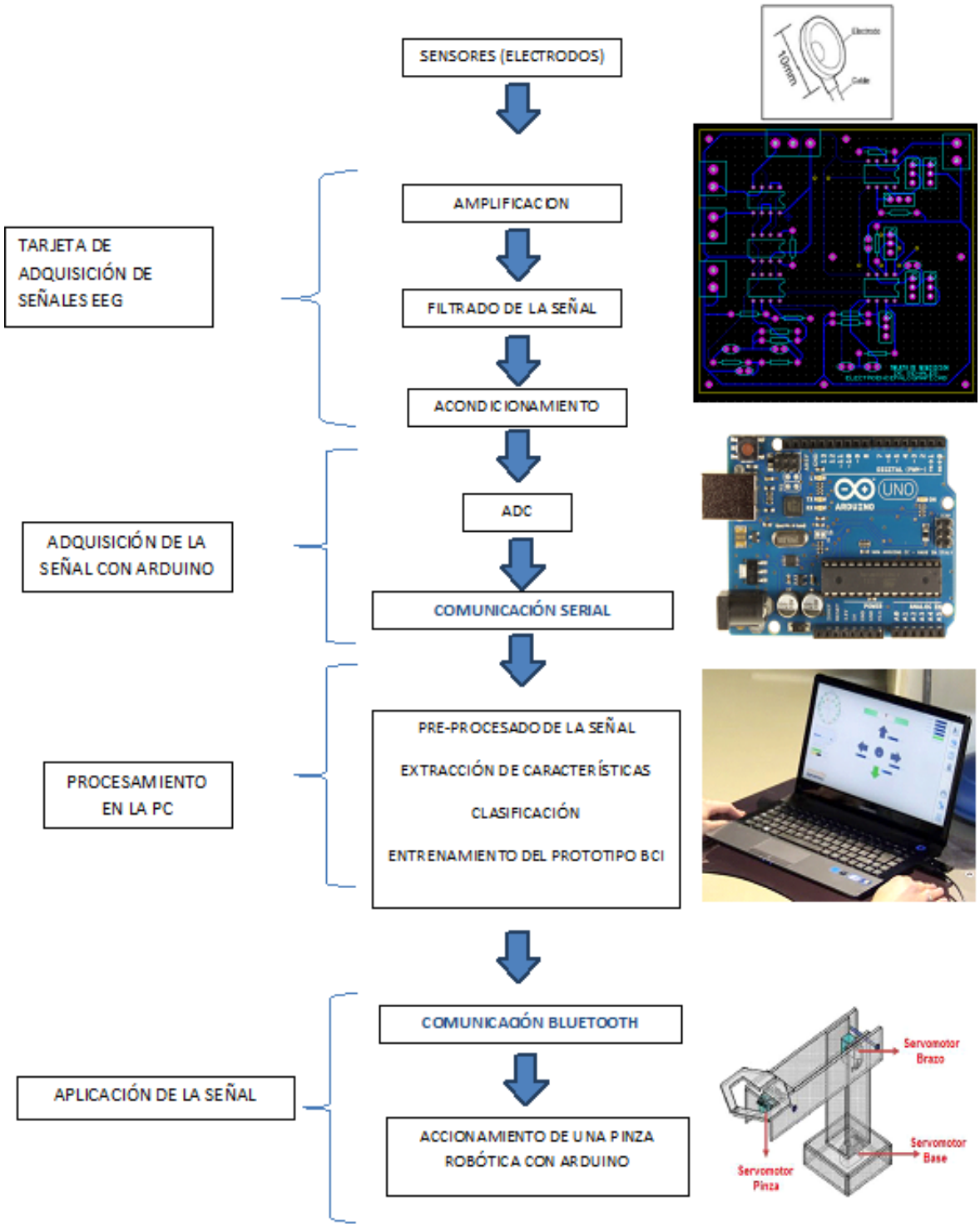

Figura 1.7: La imagen describe el diseño del prototipo de Danilo. Tomado de [9]

Figura 1.8: Sistemas comerciales BCI. A. Emotiv. B. NeuroSky. Tomado de [8]

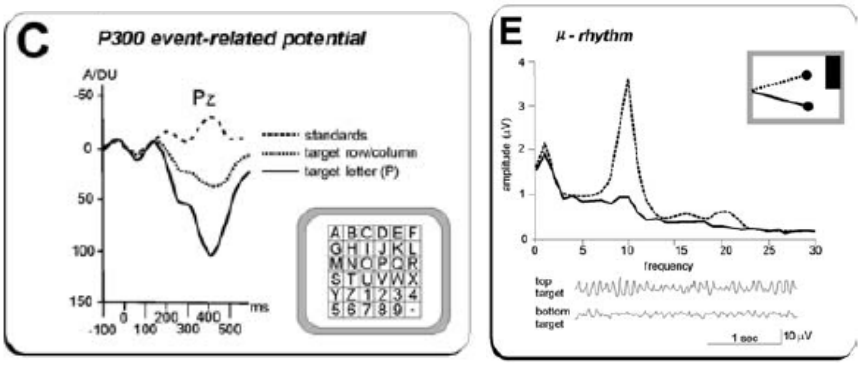

Figura 1.9: Componentes P300 y mu de un individuo. Tomada de [10]

o expresar emociones. Debido a que la mente de estos pacientes se encuentra sana es que se pueden utilizar los BCI para hacer posible que se comuniquen. Se han desarrollado distintos paradigmas, desde aplicaciones binarias de SI/NO hasta teclados virtuales para deletrear palabras.

1.3.2.2. Control de entornos y Realidad Virtual

Los entornos de realidad virtual en el entrenamiento de los BCI ha demostrado ser eficaz debido a su grado de inmersión, motivación y entorno seguro de operación. En [11] existe un mundo virtual en el cual el usuario conduce un coche sobre una calle con semáforos. El objetivo es detener el coche cuando el semáforo cambie a rojo. El sistema se basa en el análisis del potencial evocado P300. El sistema alcanzó una tasa de acierto del 85
Kar LaFleur et al. en [12] realizaron un experimento de investigación para demostrar la habilidad de los seres humanos para controlar un dron a través de un EEG BCI en un espacio tridimencional. Los sujetos sometidos a las pruebas fueron capaces de persuadir una serie de anillos y pasar a través de ellos en un ambiente del mundo real usando sus pensamientos. El estudio fue implementado con un AR Drone cuadrúpedo de bajo costo y alta disponibilidad,

y desarrolla un framework para el desarrollo de control BCI multidimensional de robots por teleoperación. El estudio permitirá a las personas que sufran de algún tipo de parálisis recuperar un poco de la autonomía perdida debido a su condición. Por último los autores mencionan que su framework permite la expansión y valoración para el control de distancias remotas con una actuación rápida y precisa usando cuadrúpedos voladores o cualquier otra implementación de robots por teleoperación.

1.3.2.3. Arte

La Universidad del Estado de Georgia, Estados Unidos trabajó en un proyecto llamado "Neural Art". El proyecto estudió diferentes métodos de representación de las señales del cerebro como expresiones de creatividad o de entretenimiento. El programa llamado "Neural Music" traducía las señales en comandos MIDI convirtiendo la actividad cerebral en tonos musicales.

1.3.2.4. Acceso a Internet

El acceso a internet a los pacientes con discapacidades motoras está considerado como un factor importante para mejorar su calidad de vida. Por medio del acceso a internet los pacientes podría ser capaces de acceder a la educación, solicitar ayuda, etc. En la Universidad de Tübingen se realizaron distintos estudios con pacientes con ELA. En dichos estudios se observó que, pese a tener teclados virtuales, los pacientes dan gran importancia al acceso a Internet.

1.3.3. Emotiv Epoc

El Emotiv Epoc es un sensor no invasivo de bajo costo para la lectura de ondas cerebrales basado en las tecnología EEG que cuenta con 14 electrodos. Su utilización resulta práctica, antes de algún experimento el usuario solo debe de colocar una solución salina en cada electrodo y posteriormente colocarse la diadema en la cabeza la cual entra sin esfuerzo. Inicialmente fue creado con el propósito de ser un periférico para juegos[13] en Windows, OS X y Linux. En la figura 1.10 se muestra al Emotiv Epoc. Sus 14 electrodos se encuentran montados en una diadema inalámbrica que funciona como dispositivo transmisor cuyas señales son recibidas en un dispositivo receptor conectado al puerto serial USB de la computadora.

Características

- Electrodos: Son electrodos de contacto los cuáles se humedecen con una solución salina. Esta solución es necesaria para leer de manera eficaz las señales producidas en el cerebro. La figura 1.11 muestra mapeo de como quedan colocados los electrodos en la cabeza.
- Señales: Ofrece una frecuencia de muestreo de 128 muestras por segundo a una transmisión interna de 2.084Hz.

- Alimentación: El dispositivo cuenta con una batería de Polímero de Litio de 800mA y 3.7V. La carga se realiza mediante un conector mini USB, la carga completa puede durar hasta 12 horas, la tabla 1.1 muestra todas las características del Emotiv Epoc. En la diadema se encuentra un led que indica el nivel de la batería durante la carga, el color rojo significa que se encuentra cargándose mientras que un color verde nos indica que ha terminado de cargarse la batería.

- Comunicación: La comunicación entre la diadema y el receptor se realiza de forma encripada a través de Bluetooth. Esta característica hace destacar al Epoc entre sus competidores.

	Diadema EEG
Número de Canales	14
Nombres de los Canales	AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4
Método de Muestreo	Muestreo secuencial, conversión analógica-digital (ADC) simple
Ratio de muestreo	128 muestras por segundo (2048 Hz internamente)
Resolución	14 bits 1 bit menos significativo = 0.51 microVolts (16 bits ADC, 2 bits para descartar ruido)
Ancho de Banda	0.2 - 45Hz, filtro digital a 50Hz y 60Hz
Filtrado	Construido en Sinc de 5th digital de filtro
Rango Dinámico	8400 microVolts (pp)
Modo de Acoplamiento	Acoplamiento AC
Conectividad	Inalámbrico propietario a 2.4GHz
Batería	LiPoly
Vida de la Batería	12 horas
Medición de la Impedancia	Calidad en tiempo real usando el sistema patentado

Cuadro 1.1: La tabla muestra las características hardware del Emotiv Epoc. Parcialmente tomada de [14]

Aunque originalmente el Epoc se creó para los juegos de computadora, la empresa Emotiv lanzó una "Edición de Investigación" del Epoc. Esta nueva edición permitió el acceso a los datos para su análisis abriendo nuevas posibilidades a la ciencia para realizar nuevos experimentos o integrarlo a los ya existentes.

En 2011 Kirill Stytsenko[15] et al. realizaron un análisis del Epoc para la CogSci Conference realizada en Liubliana, Eslovenia en 2011. Dividieron el proyecto en 4 etapas donde compararon al Emotiv EPOC y al G-TEC[16].

1. Sistemas Ejecutándose: Ejecutar simultáneamente ambos sistemas mientras sus electrodos eran colocados en las mismas áreas corticales.

2. Comparación visual: Comparando las gráficas generadas por las lecturas de cada dispositivo.

Figura 1.10: Diadema Emotiv Epoc

3. Comparación cuantitativa: Analizando y comparando los datos de cada sistema.

4. Definición de las diferencias de las señales: El G-TEC con 6 electrodos y el Epoc con 14 electrodos fueron montados en la cabeza de un sujeto para grabar su actividad cortical. Los datos grabados fueron analizados con MATLAB.

La comparación de ambos dispositivos sugirió similitud en general en los datos, sin embargo, la señal fue más limpia y fuerte en el G-TEC.
Uno de los desafíos encontrados para la realización de proyecto fue la creación del software de grabación para ambos dispositivos.

1.3.3.1. Kit de Desarrollo de Software (SDK) para la Edición de Investigación

El Emotiv Epoc cuenta con seis Kits de Desarrollo de Software o SDK por sus siglas en inglés "software development kit" que permiten el control de la API[6] y bibliotecas de la diadema.
El SDK de Investigación de Emotiv[17] es un kit de desarrollo de licencia única por usuario para empresas y escuelas que están creando aplicaciones propietarias o desarrollando nuevas aplicaciones usando los datos proporcionados por el EEG Emotiv Epoc. El SDK incluye adquisición de señales neuronales de alta resolución, procesamiento inalámbrico con la diadema y un kit de herramientas de software propietario que exponen las APIs y bibliotecas de detección del Emotiv Epoc. La interfaz incluida en el kit de desarrollo (figura 1.12) cuenta con 3 módulos que interpretan las señales obtenidas:

- $Affectiv^{TM} Suite$: Monitorea en tiempo real el estado emocional del usuario.
- $Cognitiv^{TM} Suite$: Lee e interpreta los pensamientos e intenciones concientes del usuario.

[6]Una API o Interfaz de Programación de Aplicaciones es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción

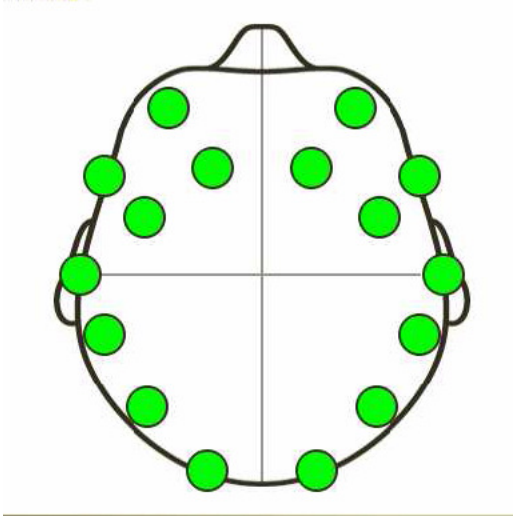

Figura 1.11: La imagen muestra como quedan los 14 electrodos colocados en la cabeza del usuario

- $Expressiv^{TM} Suite$: Utiliza las señales emitidas por la diadema para interpretar las expresiones faciales del usuario en tiempo real.

1.3.3.2. Trabajos previos con el Emotiv Epoc

En "EPOC-alypse Mind Controlled Car"[19] plantean la construcción de de un carro de control remoto que es controlado por la mente usando el Emotiv EPOC. El proyecto fue desarrollado utilizando el SDK oficial del Emotiv EPOC.
Asim Raza en "SSVEP based EEG Interface for Google Street View Navigation"[20] analiza los sistemas BCI y su aplicación en el mundo real. También desarrolla un prototipo interactivo que pueda ser controlado en un ambiente controlado para demostrar el funcionamiento de los sistemas BCI. Para el desarrollo decidió utilizar el software libre OpenViBE para la adquisición y procesamiento de las señales.
J. Castillo en [21] presenta una metodología para optimizar el diseño de una Interfaz Cerebro Computadora basada en imaginación motora. Su protocolo experimental permite la adquisición de señales EEG para 4 tareas de imaginación motora. Ocho voluntarios sanos con edades entre 22 y 28 años participaron en el experimento y suscribieron un acuerdo de consentimiento libre esclarecido, que fue aprobado por el comité de la ética UFES. Las señales fueron adquiridas con el Emtiv Epoc usando 14 electrdos, ubicados de acuerdo al sistema internacional 10/20, los canales fueron: AF3, AF4, F3, F4, F7, F8, FC5, FC6, P7, P8, T7, T8, O1 Y O2. Las señales EEG fueron muestreadas a 128 Hz, con una resolución de bit de 0.51 microvolts. Los voluntarios fueron instruidos para estar sentados con sus manos sobre las piernas, observando al centro de una pantalla de una computadora. En esta aparecía un texto con la palabra "Relax" y durante este periodo de tiempo el usuario está lo más relajado posible,

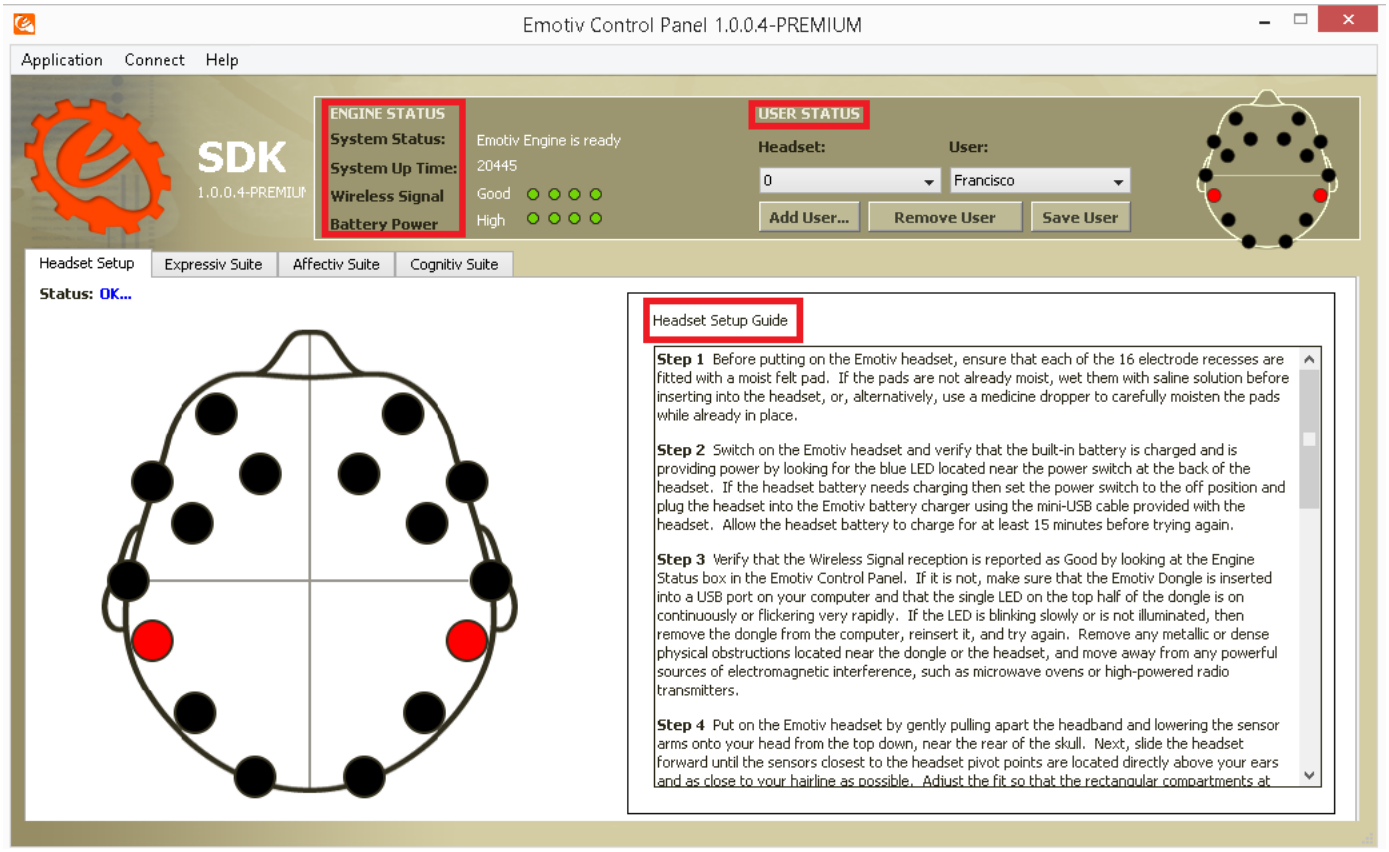

Figura 1.12: Interfaz incluida en el Emotiv SDK Research. Tomada de [18]

para reducir artefactos. Luegos de 10 segundos, aparecía otro texto, “Mental Task”, indicando el inicio de la tarea mental. Finalizando estos 10 segundos, aparecía nuevamente el texto “Relax” para indicar que la tarea había terminado. Cada sesión consistía en 20 repeticiones de la tarea mental y 20 repeticiones de descanso. Las tareas definidas fueron:

- Relajación: En esta tarea el usuario busca entrar en un estado de relajación.
- Imaginación motora del brazo izquierdo, derecho y de los dos brazos al mismo tiempo: El usuario es instruido para imaginar el movimiento del brazo o brazos, haciendo una flexión o extensión durante 10 segundos.

Muñoz Sánchez en [18] describe el proceso de la construcción de una Interfaz Cerebro Computadora con unas prestaciones superiores a las que posee la interfaz de bajo costo de la cual parten, el Emotiv Epoc.
Jorge Ierache[22] et all. utilizaron el Emotiv Epoc para desarrollar un sistema de captura y lectura de registros de estado emocional de un individuo. La aplicación que desarrollan permite la captura y almacenamiento de las señales de emociones leídas por el Emtiv Epoc durante un periodo de tiempo en el que el usuario se encuentra bajo un estímulo visual y auditivo. Para realizar la lectura utilizan el SDK oficial del Emotiv Epoc en cual se encarga

de detectar las emociones, posteriormente, por medio de la API de Emotiv se conecta su aplicación. Su aplicación fue desarrollada en .NET C# y SQLite. En la figura 1.13 se puede observar un esquema de como funciona el sistema desarrollado.

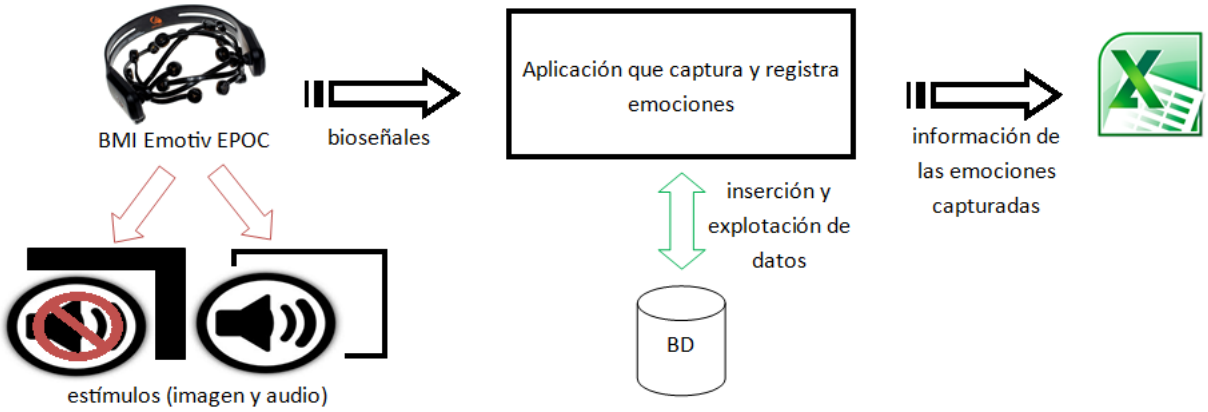

Figura 1.13: Integración de Emotiv y la aplicación de captura y registro de emociones. Tomada de [22]

1.3.4. Emokit, una alternativa libre

Una desventaja para los desarrolladores al utilizar el Emotiv Epoc es que las herramientas de desarrollo que ofrece Emotiv son propietarias y de código cerrado. Esto significa que un desarrollador debe de pagar una licencia para poder hacer uso de dichas herramientas. Afortunadamente un grupo de desarrolladores han creado un proyecto de código abierto llamado Emokit que permite hacer uso de la diadema Emotiv Epoc sin hacer uso de las herramientas proporcionadas por Emotiv[23].La librería de Emokit fue desarrollada por Cody Brocious y Kyle Machulis. Emokit es un controlador que expone la información enviada por la diadema a través de su transmisor USB.

1.3.4.1. Emotiv Visualizer

Uku Loskit en [23] utiliza los datos obtenidos con el software Emokit para ser visualizados en su nuevo software. Este software, de nombre Emotiv Visualizer fue desarrollado en Python, complementado por las librerías numpy, matplotlib y Tkinter.
La figura 1.14 muestra la interfaz resultante de Uku. Las gráficas en la izquierda representan los 14 electrodos en tiempo real. La imagen de la derecha muestra la intensidad de la señal en tiempo real. El color de los electrodos indica la calidad de la señal de cualquier punto en tiempo, así el usuario puede acomodar la diadema para que los electrodos proporciones una mejor calidad en las lecturas. El rango de colores de los electrodos son de negro (sin señal) a rojo y de amarillo a verde(señal perfecta).

El Emotiv Visualizer realiza las lecturas del Emokit y la graficación de estas en la misma interfaz. Esto representa un problema ya que al ser el funcionamiento secuencial no refleja un panorama real de los datos que se están obteniendo del Emokit el cual publica aproximadamente 15 lecturas por segundo. El modo de trabajo secuencial también ocasiona que la memoria RAM del equipo ejecutando el software se sature causando que la interfaz y la computadora se congelen. En algunos casos hubo incluso que forzar el apagado de la computadora para poder continuar realizando los experimentos.
El Emotiv Visualizer, además, no permite la interacción del usuario con la interfaz, esto ocasiona que cada vez que se termine de registrar un set de datos en usuario debe de cerrar la interfaz y volverla a abrir para realizar otro experimento. Otro inconveniente del Visualizer es que no guarda un registro de las señales leídas mientras se ejecutó.
Estas desventajas llevaron a plantearse la idea de realizar un nuevo software para solucionar dichos problemas. De esto nace la idea de realizar el software presentado en esta tesis el cual realiza la lectura/graficación en tiempo real, cuenta con tres botones para controlar la visualización de los datos (reproducir, pausar, detener) y muestra un registro de las señales leídas que al detener la prueba guardará en un archivo en disco duro CSV. Todos los detalles de la interacción y funcionamiento de la nueva interfaz serán explicados más adelante.

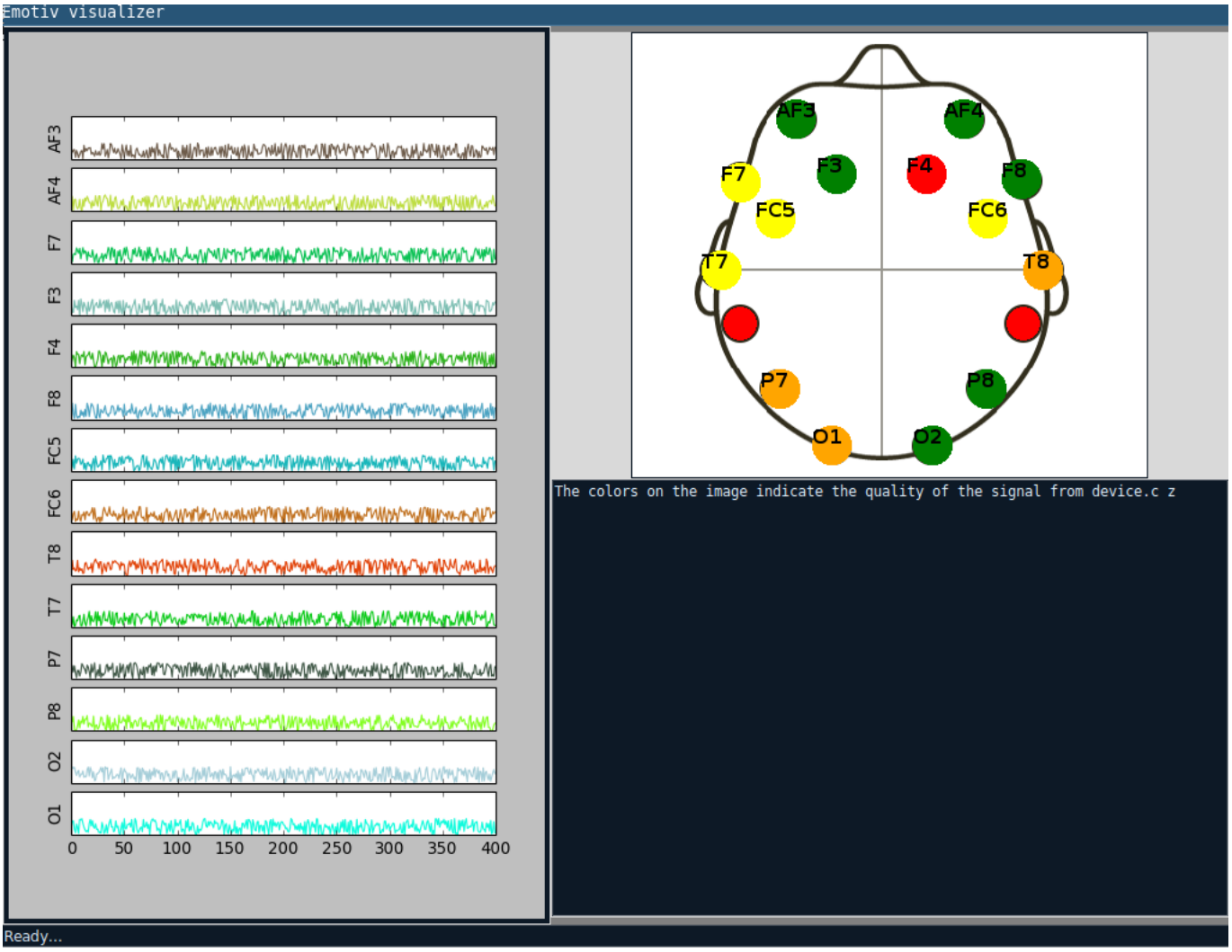

Figura 1.14: Emotiv Visualizer. La columna de la izquierda muestra la graficación de las 14 señales recibidas. La imagen de la parte superior derecha muestra un mapeo de la colocación de los electrodos, un color rojo significa que no esta leyendo y un color verde significa que esta leyendo correctamente. La imagen inferior derecha muestra un mensaje explicando los el esquema de colores.Tomado de [23]

Capítulo 2

Diseño

La Universidad Veracruzana adquirió un Emotiv EPOC en 2014 para el laboratorio de robótica donde los alumnos de la Facultad de Ingeniería participan en diferentes proyectos tecnológicos. Este trabajo de tesis contribuye a un proyecto donde se controlará un robot por medio de la mente a través del Emotiv EPOC. Con el proyecto resultante de este trabajo se tendrá una interfaz la cual graficará las señales del Emotiv EPOC además de tener la capacidad de grabar los datos obtenidos cada vez que se ejecute un "experimento" obteniendo un archivo separado por comas.
Al momento de la investigación se encontraron algunos programas que mostraban la información obtenida de forma gráfica, sin embargo el principal inconveniente de estos era el desfase entre los datos que se estaban obteniendo y su graficación.
Un ejemplo de ello es el antes mencionado Emoitv Visualiser. Para atacar ese problema en este proyecto se utilizó un framework llamado ROS. En este proyecto se utilizó ROS para crear un escenario de comunicación en tiempo real, donde el Emokit[1] publica a través de mensajes los datos obtenidos del Epoc así mismo el nuevo software leerá los mensajes publicados en ROS para obtener los datos para graficarlos y generar el "log", todo esto ya en tiempo real.

2.1. Robot Operating System (ROS)

ROS (Robot Operating System)[24] provee librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes y más. ROS está bajo la licencia open source, BSD.
En ROS: an open-source Robot Operating System[25] se explica el uso de la plataforma ROS para el desarrollo de aplicaciones de robótica y resumen los objetivos filosóficos de ROS en:

- Per-to-per
- Basado en herramientas

- Multilenguaje
- Ligero
- Gratuito y de Código Abierto

Para entender mejor como la interfaz funciona e interactúa con el emokit a través de ROS hay que explicar los términos Topics, mensajes, publisher y subscriber. ROS trabaja como un servidor el cual se ejecuta para que se puedan establecer todas las comunicaciones entre los procesos, dentro de una interacción de procesos intervienen tres factores:

- Mensajes: Son archivos donde se definen que datos y de que tipo serán comunicados por medio de el, cada archivo es un paquete que contiene diferentes tipos y nombres de variables.
- Publishers: Son los programas o procesos que se encargarán de publicar información en los mensajes para que estos sean leídos por uno o varios subcribers.
- Subcribers: Son los programas o procesos que leen los mensajes publicados por los publishers.

Este esquema de funcionamiento es fácilmente comparable con el clásico esquema de comunicación del habla donde contamos con un emisor(publisher), un mensaje y uno o varios receptores(subcribers).

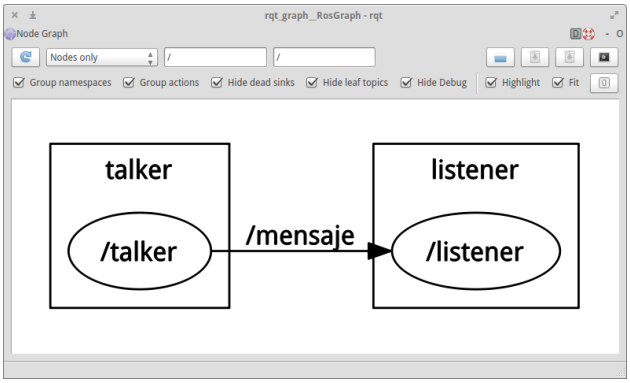

Figura 2.1: Comunicación ROS. Una aplicación “talker” publica en una aplicación “listener” por medio de un mensaje.

ROS le permite al software resultante en este trabajo comunicarse con cualquier programa que se suscriba a nustro mensaje quedando no solo limitado a trabajar con el emokit. El software solo lee los mensajes publicados por ROS independientemente de quien los esté enviando. Podemos ver esto como una ventaja ya que permite que el software se use en conjunto con cualquier otro programa o proceso que publique el mensaje que el software lee.
En Things that twitter: social networks and the internet of things[26] utilizan ROS aplicado

en las redes sociales. ROS permite intercambiar información por medio de servicios con mensaje de request y response definidos. La información es intercambiada por una arquitectura publish/suscribe donde los procesos permiten que sus datos estén disponibles para que otros procesos puedan utilizarlos.

2.1.1. Estructura de un proyecto en ROS

En proyecto es ROS es creado por el comando "roscreate-pkg nombre_proyecto dependencia1 dependencia2"[1]. Este comando crea una estructura de carpetas como se muestra en la figura 2.2, ROS utiliza esta estructura para funcionar correctamente. Para este proyecto se utilizaron las carpetas msg donde se crean las definiciones de los mensajes; scripts donde se guardan todos los códigos fuente; y img que es una carpeta creada adicionalmente para almacenar las imágenes de los botones.

2.2. Desarrollo

El software se desarrolló en Python, utilizando las librerías PyQt y matplotlib; ejecutándose sobre el framework ROS. Python es un lenguaje de programación interpretado, interactivo y orientado a objetos que provee estructuras de datos de alto nivel como listas y diccionarios, módulos, clases, excepciones, manejo automático de memoria, etc. Además es un lenguaje poderoso y multi propósito que cuenta con una sintaxis simple y elegante[27]. Para la creación de la interfaz se decidió utilizar el framework QT integrado a Python mediante PyQt. PyQt un binding[2] de QT para Python v2 y v3 que puede ser ejecutado en todas las plataformas soportadas por QT incluyendo Windows, MacOS/X y Linux. Está implementada por medio de módulos de Python contenidos en más de 620 clases. Matplotlib es una librería de Python para graficación 2D que produce figuras de calidad en una variedad de formatos y entornos interactivos a través de distintas plataformas.
El software contará con una interfaz sencilla, 14 gráficas donde se mostrarán cada una de las señales recibidas; tres botones para reproducir, pausar y detener la graficación de las señales y un cuadro de texto para escribir el nombre con el que se generará el archivo de registro y un campo de texto que servirá para visualizar cada paquete de señales que se va recibiendo. Cada gráfica corresponde a una señal del Emotiv Epoc. Las gráficas se refrescan cada 30 señales recibidas esto limitado principalmente por el tamaño de la interfaz. En el cuadro de texto se mostrará el log, este log muestra los paquetes de señales recibidas, es decir, cada vez que se recibe un mensaje se mostrarán las 14 señales como una lista en el cuadro de texto y se dibujará un punto nuevo en cada una de las 14 gráficas. Por último los botones controlan el dibujado en la gráfica y en el cuadro log. Cabe mencionar que el software siempre estará leyendo los mensajes publicados en ROS independientemente de que se encuentre graficando o no, los botones, por lo tanto, controlan lo que el usuario necesita ver y registrar

[1] Creating a ROS Package: `http://wiki.ros.org/ROS/Tutorials/CreatingPackage`

[2] Adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquél en el que ha sido escrita.

al momento que está utilizando la interfaz. En la figura 2.3 se ilustra como se pretende que luzca la interfaz.

2.3. Diagramas

A continuación se presentan 3 diagramas para explicar la construcción e interacción del nuevo software. El primero es un diagrama de secuencia donde se muestra la interacción entre los módulos y programas implicados en el software. El segundo es un diagrama de componentes muestra de la interacción entre las clases y métodos. Por último se presenta un diagrama de clases donde se definen los objetos y métodos que contiene cada clase de la interfaz creada, en este diagrama ya no se toman en cuenta las clases de los programas con los que interactúa.

2.3.1. Diagrama de Secuencia

El diagrama de secuencia presentado en la figura 2.4 ilustra las fases y estados por los que pasa el software desde que el usuario comienza a usar el sistema. Para el uso de la interfaz intervienen 3 componentes, el primero es la interfaz en sí, el segundo es el framework ROS ejecutándose en segundo plano y el tercero es la adaptación del Emokit a ROS la cuál se encuentra publicando las señales leídas del Epoc a un mensaje de ROS. Como se puede observar en el diagrama por un lado se encuentra el Emokit únicamente publicando el mensaje, y por otro lado está la interacción del usuario con la interfaz, esta interacción es la que interesa explicar. El siguiente pseudocódigo explica de manera general al diagrama de secuencia.

1. El usuario ejecuta la interfaz.
2. La interfaz ejecuta su método listener() con el que se conecta a ROS.
3. Ya que se encuentra conectado, el método callback() se encontrará siempre conectado a ROS. A través de este se lee el mensaje publicado por Emokit.
4. La interacción con la interfaz sucede por eventos a través de los botones.
   a) El botonEjecutar() al ser presionado manda a llamar un método que deshabilita la graficación.
   b) El botonPausar() deshabilita únicamente deshabilita la graficación sin limpiar las gráficas ni el cuadro de log.
   c) El botonDetener() deshabilita la graficación, limpia las gráficas y el cuadro de log y además manda a llamar al método generarLog() que generará un archivo CSV[3] validando que exista un nombre ingresado en el cuadro de texto correspondiente o de lo contrario mandar un mensaje para ingresarlo.

[3]Archivo Separado por Comas

2.3.2. Diagrama de Componentes

En este se muestran las clases y métodos que intervienen únicamente en el funcionamiento de la interfaz. Las clases se encuentran distribuidas en tres archivos de python, el archivo interfaz.py cuenta con la clase GUIForm que contiene toda la lógica en interacciones; el archivo GUI.py contiene a la clase Ui_EpocGUI que define los componentes de la interfaz en PyQt y por último el archivo matplotlibwidgetFile.py contiene a las clases MplCanvas y matplotlibWidget donde se define la plantilla de las gráficas de matplotlib. En el diagrama de componentes de la figura 2.5 las clases de matplotlib se representan con su nombre de archivo ya que ambas son dependientes y sirven para un único fin.
Como se puede observar en el diagrama al lanzar la aplicación se hace una llamada a la clase GUIForm esta crea un objeto de la clase Ui_EpocGUI para cargar la interfaz Qt. La clase Ui_EpocGUI crea 14 objetos de la clase matplotlibWidget para cargar la platilla de la gráfica e integrarla a PyQt por medio de un elemento de tipo widget. Después de esto la interfaz manda a llamar a su método listener() que a su vez manda a llamar al método callback() para conectarse a ROS y mantenerse escuchando los mensajes que se publiquen en este.

2.3.3. Diagrama de Clases

En el diagrama de clase de la figura 2.6 podemos observar con mas detalle lo descrito en el diagrama de componentes. La clase GUIForm es el programa principal donde se tiene toda la lógica y se crean los objetos de las otras clases. La clase Ui_EpocGUI es la definición de la interfaz y las clases MplCanvas y matplotlibWidget definen la gráfica de matplotlib.

2.3.3.1. Clase GUIForm

Variables y Objetos

- self.ui: Objeto de la clase Ui_EpocGUI, mediante este se crea, lanza y utilizan los elementos de la interfaz Qt
- self.grafica: Con esta variable boleana controlaremos el dibujado de las gráficas.
- self.frecuencias[]: Lista donde se guardan las señales obtenidas del mensaje de ROS. Se guardan las señales en una lista para poder mandarlas como parámetros a las funciones de graficación.

Métodos

- __init__(): Inicializa todas las variables.
- graficar(): Mediante 14 instrucciones self.ui.widget.canvas.ax.plot(frecuencias[0]) incluye los puntos a la grafica, self.ui.widget.canvas.draw() permite que los puntos añadidos sean visibles.

- habilitaGraficar(): Vuelve “verdadera” a la bandera self.grafica.
- pausaGrafcar(): Vuelve “falsa” a la bandera self.grafica.
- detenerGraficar(): Vuelve “falsa” a la bandera self.grafica y llama al método generarLog().
- generarLog(): Extrae el texto del cuadro de log para guardarlo en un archivo CSV.
- callback(): Función especial de ROS que se conecta al servidor y se encuentra siempre leyendo los mensajes que se publiquen en el.
- listener(): Función de ROS para inicializar la comunicación, manda a llamar a callback().

2.3.3.2. Clase UI_EpocGUI

Variables y Objetos

- self.widget: Objeto de la clase matplotlibWidget, este permitir mostrar las gráficas dentro de PyQt.
- self.botonPausar, self.botonReproducir, self.botonDetener: Definición de los botones.
- self.textBrowser: Objeto para el cuadro de texto de log.
- self.textNombrePrueba: Objeto para el cuadro texto para ingresas el nombre de la prueba que será parte del nombre del archivo generado.

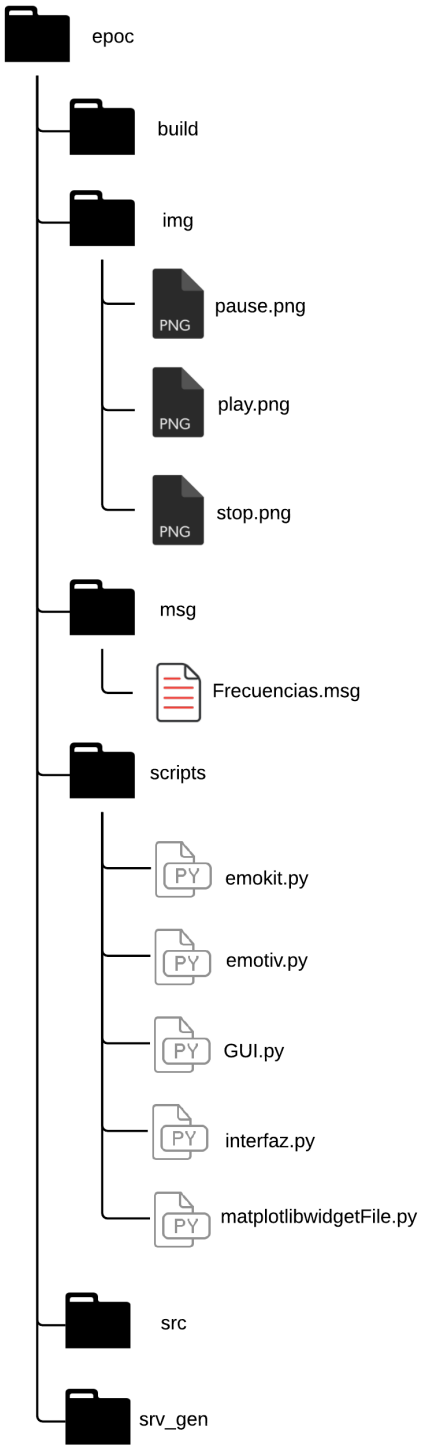

Figura 2.2: Estructura de carpetas generada para un proyecto en ROS. Se ilustran los archivos generados para el proyecto.

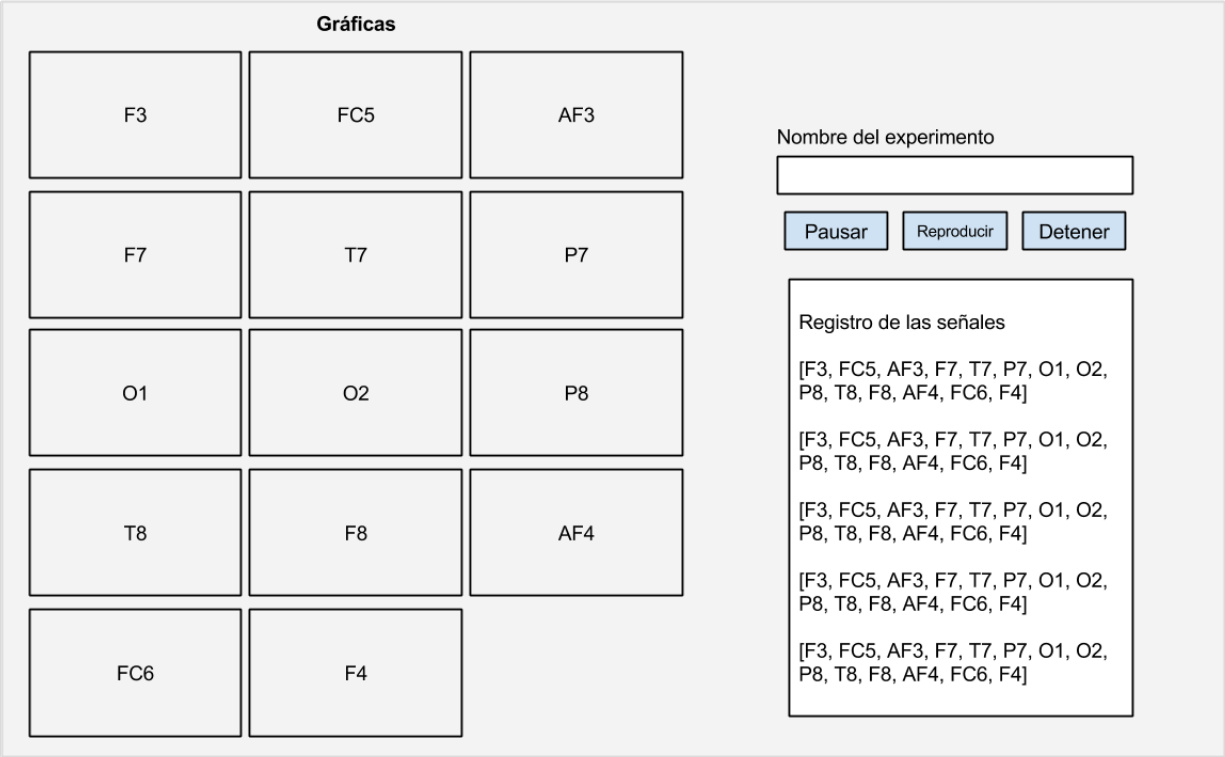

Figura 2.3: Interfaz. Los 14 cuadros de la izquierda son gráficas que corresponden a cada una de las señales leídas. En la parte superior derecha se encuentran los controles de la interfaz y el espacio para escribir el nombre que se le dará a la sesión de datos leídos. El cuadro inferior derecho mostrará el log.

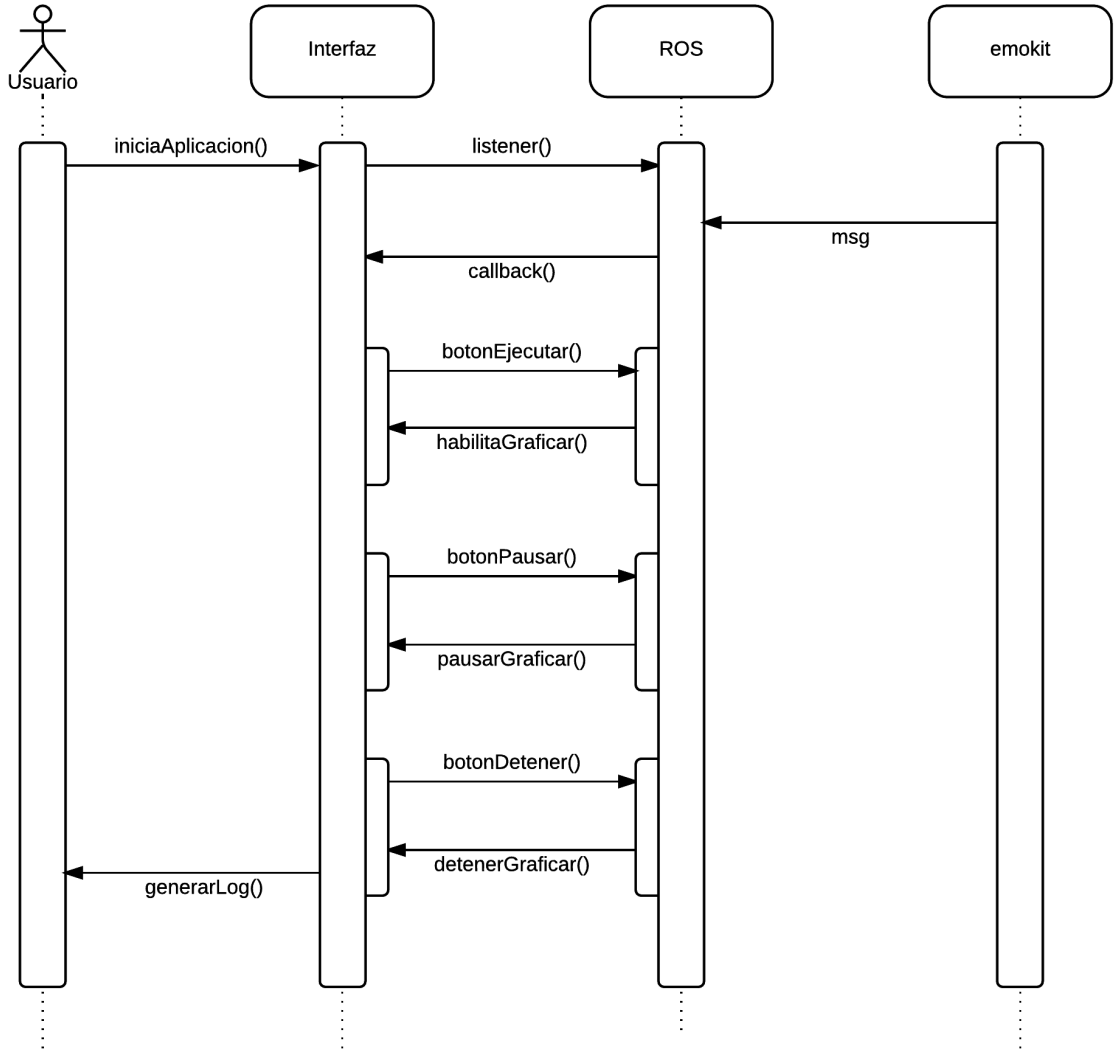

Figura 2.4: Diagrama de Secuencia. El diagrama muestra de manera general los estados por los que pasa el nuevo software

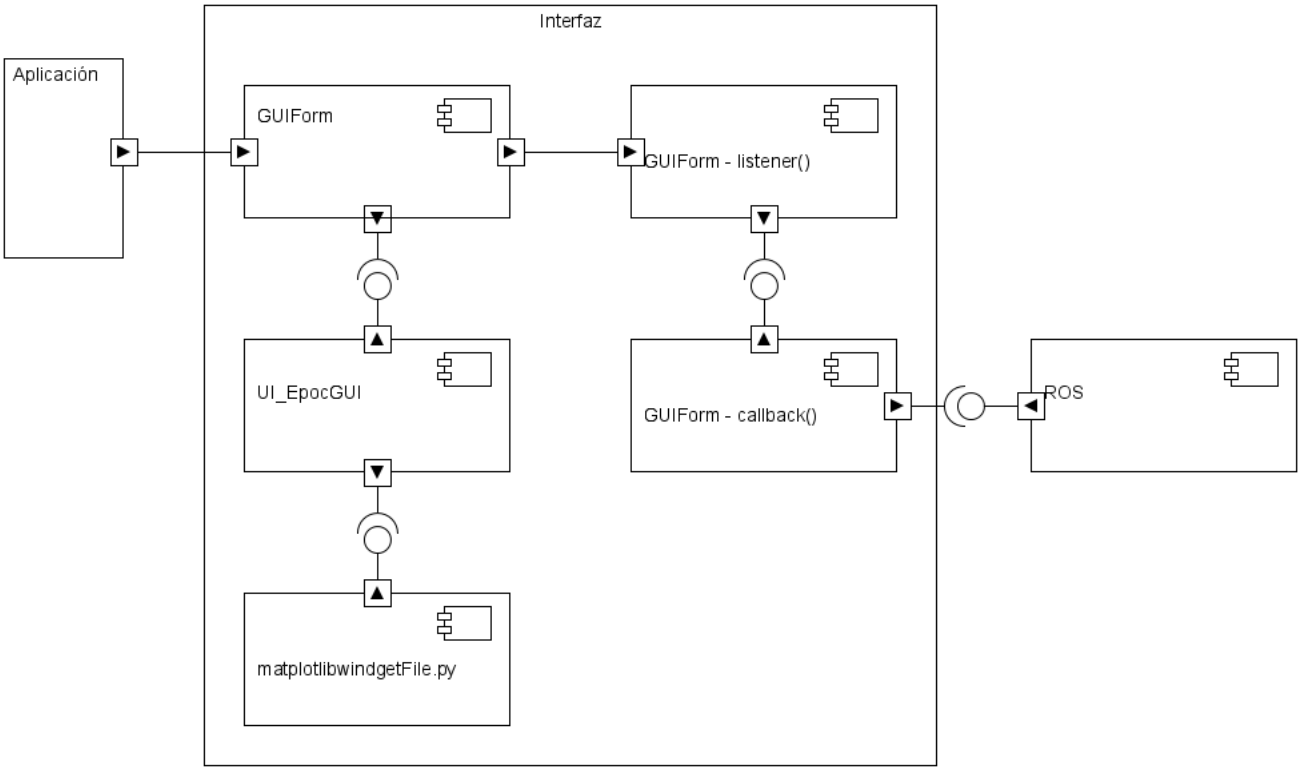

Figura 2.5: Diagrama de Componentes. Interacción entre los módulos de la interfaz.

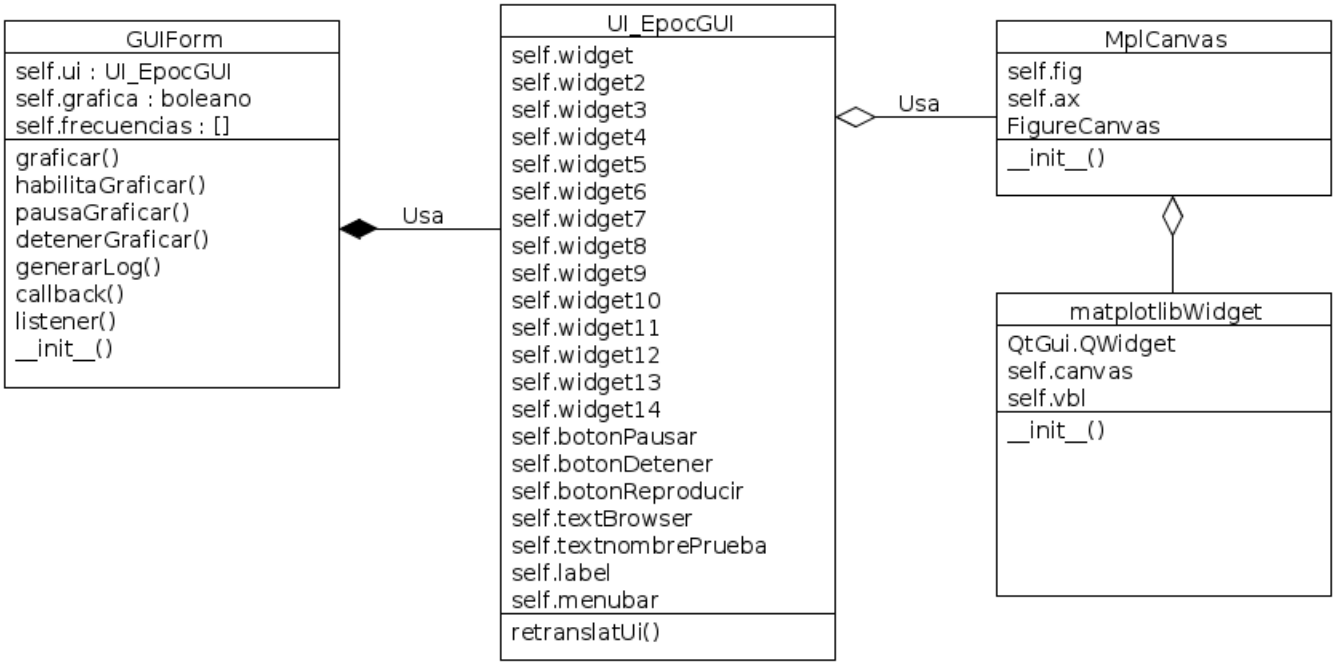

Figura 2.6: Diagrama de Clases. Definición de las clases de la interfaz así como sus objetos y métodos.

Capítulo 3

Implementación y Resultados

3.1. Implementación

La implementación del software desarrollado en este proyecto requiere de la instalación y configuración de las tecnologías explicadas anteriormente. Las instrucciones y configuración que a continuación se describirán son para sistemas operativos basados en Linux/Debian. El proyecto se probó en los sistemas operativos Ubuntu 12.04, Ubuntu 14.04, ElementaryOS Freya y ElementaryOS Luna, todos estos basados en Linux/Debian.

3.1.1. Instalaciones y Configuraciones Previas

Para el funcionamiento del nuevo software es necesario:

- Instalar Git: En Ubuntu basta con abrir la terminal y ejecutar el comando "sudo apt-get install git".
- Instalación de Emokit: `https://github.com/openyou/emokit`
- Instalación y Configuración de ROS: Dependiendo de la versión de Ubuntu se recomienda
  - Ubuntu 12.04, 12.10, 13.04, Windows, OS X, : ROS Hydro `http://wiki.ros.org/hydro/Installation`
  - Ubuntu 13.10, 14.04, OS X, Android, Arch Linux: ROS Indigo `http://wiki.ros.org/indigo/Installation`
  - Ubuntu 14.10, 15.04, OS X, Android: ROS Jade `http://wiki.ros.org/jade/Installation`

  Para ver todas las versiones y plataformas de ROS consultar: `http://www.ros.org/reps/rep-0003.html` y `http://wiki.ros.org/Distributions`. Para la configuración de ROS consultar `http://wiki.ros.org/ROS/Tutorials`, es muy importante realizar correctamente la configuración de las variables de entorno.

- Instalación de Python: `https://www.python.org/downloads/`. Por motivos de compatibilidad este proyecto funciona con la versión de Python 2.7. En el caso de Ubuntu esta versión ya viene instalada en el sistema operativo por defecto.
- Instalación de librerías QT y PyQT: `http://www.riverbankcomputing.co.uk/software/pyqt/download`. El proyecto funciona con PyQt4, para Linux se recomienda instalarlo desde el Centro de Software (Ubuntu), Synaptic o ingresando "sudo apt-get install python-qt4" en la terminal (sin comillas).
- matplotlib: `http://matplotlib.org/downloads.html`. Para Linux utilizar Synaptic e instalar el paquete "python-matplotlib" o ingresar en la terminal "sudo apt-get install python-matplotlib" sin comillas.

3.1.2. Ejecución del Software

Para instalar y ejecutar el software se requiere (Las instrucciones indicadas están hechas para trabajar en Ubuntu), si ya se ha descargado el software con anterioridad se procede desde el paso 4:

1. Abrir la terminal y ejecutar el comando "roscd" que nos ubicará en el workspace de ROS.
2. Ejecutar "cd sandbox".
3. Ingresar el comando "git clone https://github.com/carlosbulnes/epoc.git". Se descargará el proyecto en la carpeta sandbox del workspace de ROS. También se puede ingresar a `https://github.com/carlosbulnes/epoc` y descargar el proyecto manualmente. Descomprimir el archivo descargado y copiarlo a "carpeta personal/workspace/sandbox" y renombrarlo la carpeta con el nombre "epoc".
4. Iniciar el servidor ROS: Para que todo pueda funcionar ROS debe de estar ejecutándose, para esto abrimos una terminal y escribimos el comando "roscore". La figura 3.1 muestra lo que nos debe aparecer si el comando funcionó correctamente.
5. Iniciar el Emokit: El emokit se inicia mediante el comando "rosrun epoc emokit.py" este debe de ejecutarse en otra terminal.
6. Iniciar la interfaz: Ingresar en una tercera terminal el comando "roscd epoc" para movernos al directorio del proyecto y posteriormente ejecutar "rosrun epoc interfaz.py". Se abrirá una ventana como se muestra en la figura 3.2.

Es importante mencionar que la carpeta del proyecto debe ser guardada en el directorio de trabajo definido en la instalación de ROS. En el caso de Ubuntu con ROS Fuerte ruta donde se debe guardar el proyecto es `/home/tu_usuario/fuerte_workspace/sandbox/` es importante también que la carpeta de este proyecto se llame "epoc".

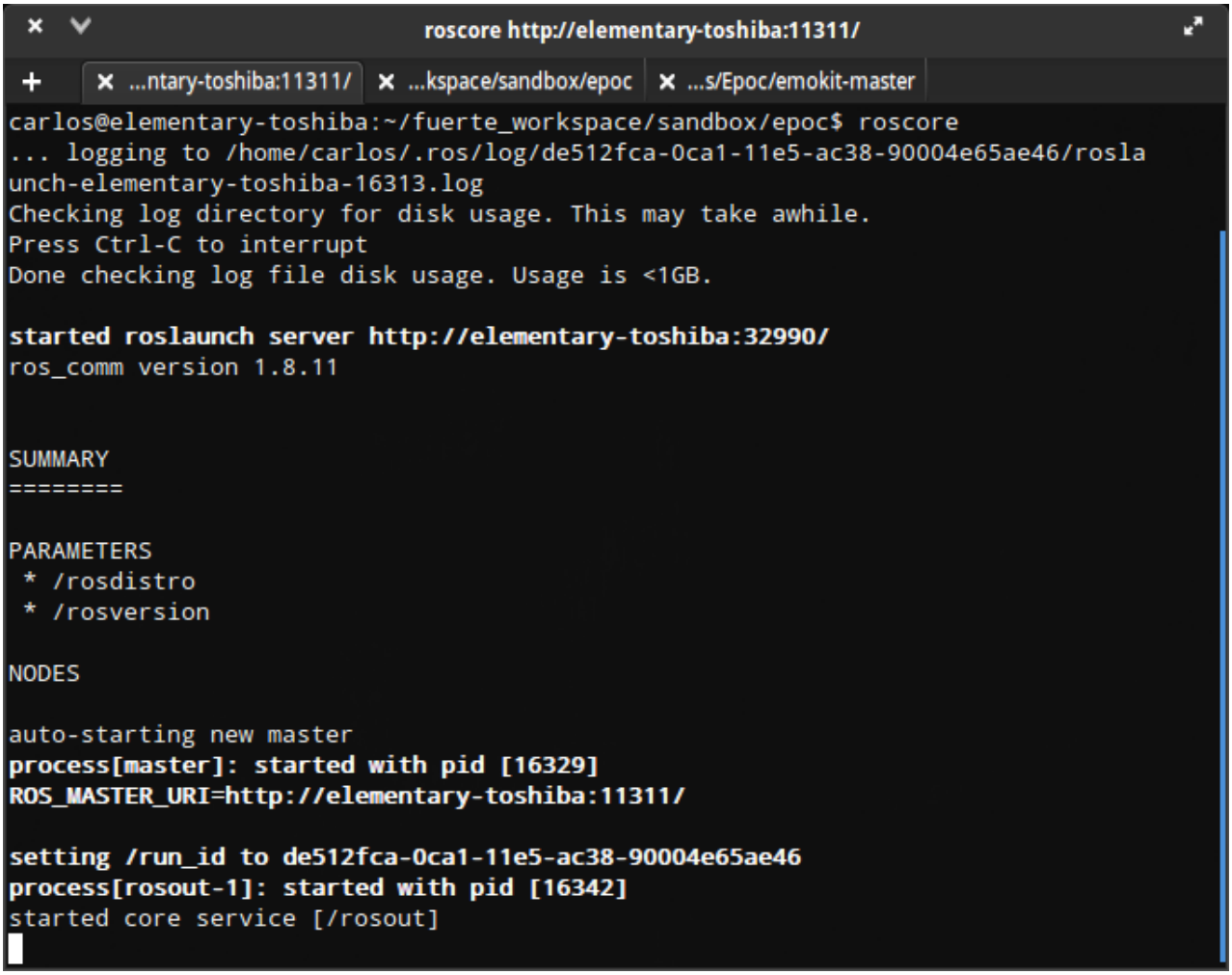

Figura 3.1: Comando “roscore” funcionando correctamente

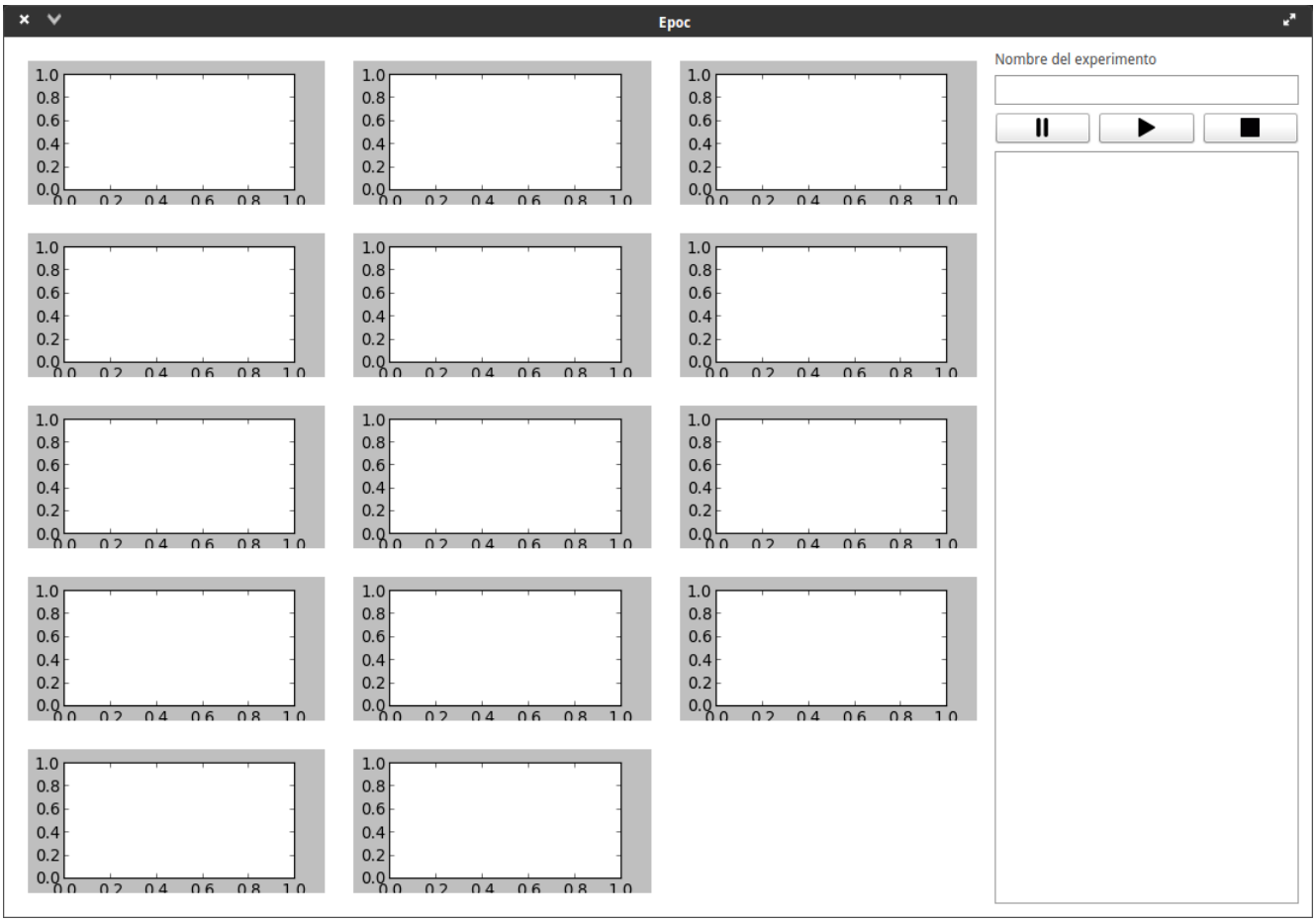

Figura 3.2: Ejemplo del nuevo software cuando se acaba de iniciar. Los 14 cuadros de la izquierda son gráficas que corresponden a cada una de las señales leídas. En la parte superior derecha se encuentran los controles de la interfaz y el espacio para escribir el nombre que se le dará a la sesión de datos leídos. El cuadro inferior derecho muestra el log.

3.1.3. Diagramas

3.1.3.1. Diagrama de Flujo

En el diagrama de flujo de la figura 3.3 se pueden apreciar las etapas que tiene el software en su primera ejecución. Ese diagrama solo muestra el primer caso, en adelante el control de la interfaz será por medio de los botones con los que cuenta. En la primera instrucción se inicializan las variables necesarias para lanzar la interfaz (self.ui), controlar la graficación (self.grafica) y almacenar las señales que sean leídas en los mensajes ROS (self.frecuencias). Posteriormente se manda a llamar al método listener() donde se llaman los métodos pertenecientes a ROS, rospy.init_node() y rospy.Suscriber, para configurar y establecer la comunicación respectivamente. Después el método listener() manda a llamar al método callback(), este método es muy importante pues se encontrará siempre presente y es por medio del cual leeremos los mensajes que se publiquen en ROS. Este mismo método se encarga de generar el log en el cuadro de texto y mandar a llamar al método graficar() que contiene las funciones necesarias para dibujar los puntos en las 14 gráficas de la interfaz.

3.1.3.2. Casos de Uso

El diagrama de la figura 3.4 ilustra las operaciones que la persona que utilice la interfaz tendrá disponibles. El usuario podrá interactuar y controlar la visualización de los datos por medio de los botones. Los botones de "Reproducir" y "Pausar" controlan la información mostrada en las gráficas y el cuadro de log, mientras que el botón "Detener" detiene la visualización, extrae la información del cuadro de log a una variable para grabarla en un archivo en disco duro CSV para que el usuario pueda consultarlo aún después de haber cerrado la interfaz. El archivo se genera con el nombre que sea ingresado en el cuadro de texto correspondiente concatenando la fecha y hora del sistema.

3.2. Resultados

3.2.1. Lectura de datos aleatorios

La primera de las pruebas se realizó publicando datos aleatorios en ROS y leyéndolos en la interfaz. Se obtuvieron 129 lecturas en un tiempo de 1 minuto aproximadamente como se ilustra en la tabla 3.1. Como podemos observar en la figura 3.5 las gráficas y el log se generaron correctamente. Por último las figuras 3.6 y 3.7 muestran que el archivo CVS se ha generado correctamente con los datos generados en la prueba.

Hora Inicial	Hora Final	Tiempo Exacto	Número de Lecturas
17:23:05	17:24:07	0:01:01	129

Cuadro 3.1: Tiempo y número de lecturas realizadas por la interfaz recibiendo datos aleatorios.

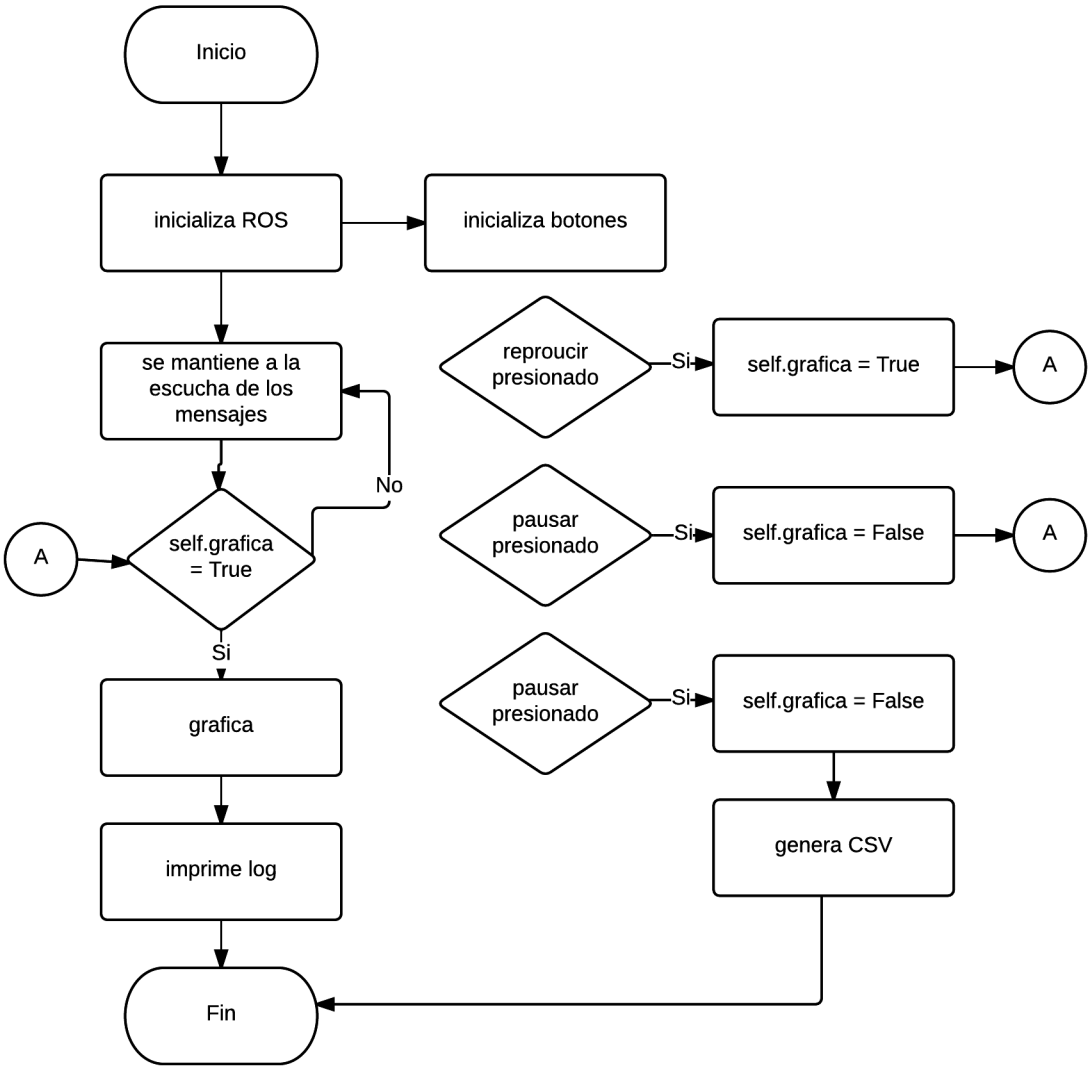

Figura 3.3: El diagrama muestra el flujo que sigue el nuevo software con respecto al orden en que se ejecutan las instrucciones.

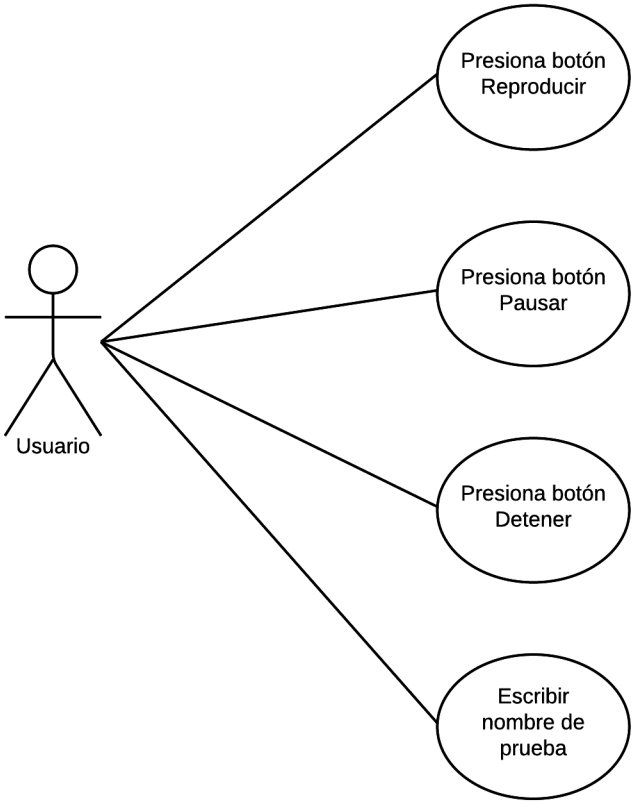

Figura 3.4: En diagrama de casos de uso ilustra la funciones que el usuario tendrá para interactuar con el nuevo software.

3.2.2. Lectura de las Ondas Cerebrales

La segunda prueba se realizó ya utilizando el Emotiv Epoc, el Emokit y la interfaz (figura 3.8). Las lecturas obtenidas por minuto fueron aproximadamente iguales a las de la prueba anterior, realizando 120 lecturas en 59 segundos (tabla 3.2). Se puede observar de igual forma en las figuras 3.9 y 3.10 que el guardado de los datos se generó correctamente. De manera adicional las figuras 3.11 y 3.12 nos muestran los datos que el Emokit está leyendo del Emotiv Epoc.

Hora Inicial	Hora Final	Tiempo Exacto	Número de Lecturas
18:52:28	18:53:28	0:00:59	120

Cuadro 3.2: Tiempo y número de lecturas realizadas por la interfaz recibiendo datos del Emokit.

Figura 3.5: Interfaz graficando las señales recibidas aleatoriamente.

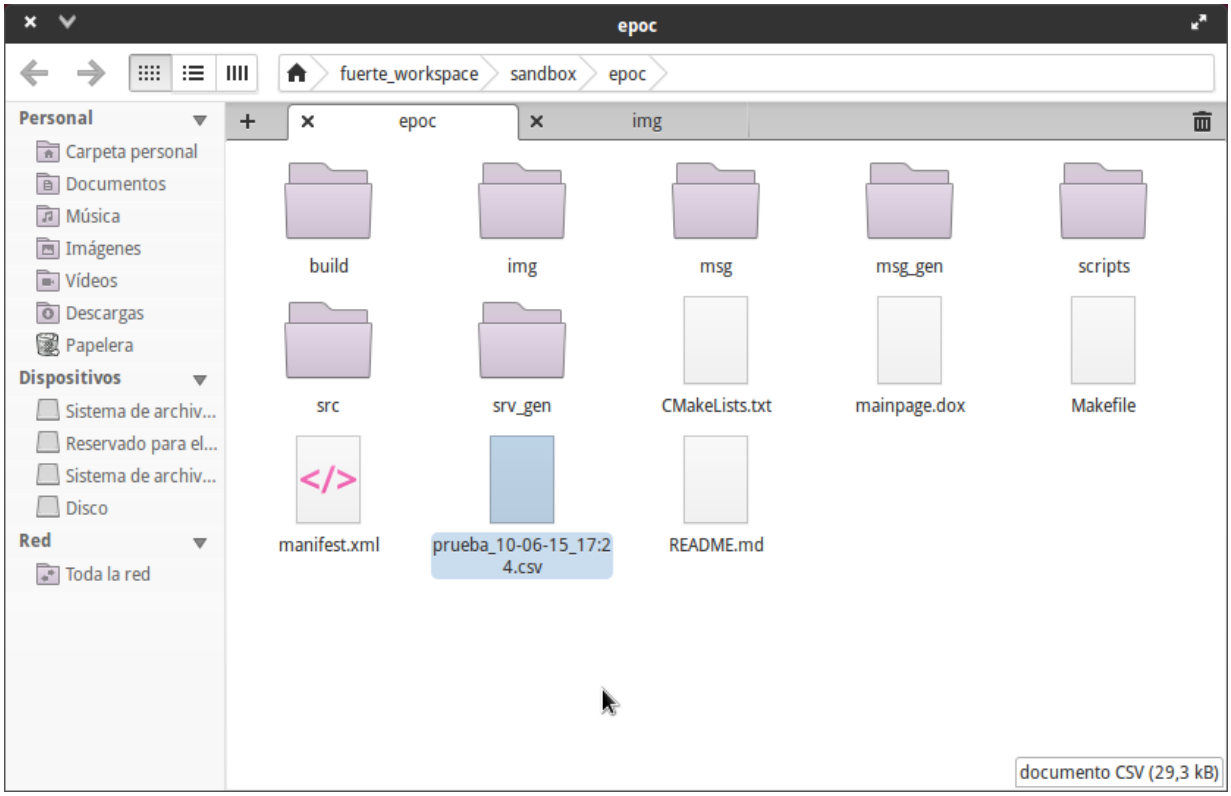

Figura 3.6: Archivo CVS creado para la prueba aleatoria.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	F3	FC5	AF3	F7	T7	P7	O1	O2	P8	T8	F8	AF4	FC6	F4
2	-0.918169348	0.8095194654	0.0275633313	0.6913182779	-1.157759686	0.5581146862	-0.117436288	-0.92687021	0.6191545442	-1.023162747	-1.086259994	0.676996821	0.8545175395	-0.179372231
3														
4	-0.506937162	0.6439613099	0.4792297467	0.637958326	-0.071039942	0.3076946151	-1.820217197	0.2256084383	-0.274217412	1.7576867809	1.2133714748	0.8233019146	-1.138139621	-0.15402018
5														
6	1.3263228843	1.6246082039	-0.582977778	1.2716501499	-0.166833581	-1.163708792	-0.235121134	1.3092104904	0.6857541507	-0.392347565	0.9416018872	-1.235146684	0.2527661961	1.8558912059
7														
8	-0.489969571	2.4675340261	0.3893168538	-0.616310258	2.4047945427	-0.658889786	-1.280869605	0.1134854394	1.7145208728	-1.501303984	0.7055777377	0.4864516753	1.131299691	0.7082624456
9														
10	-1.887731241	0.3611823613	-1.596695485	-0.035407845	-0.544025609	0.2761672675	0.0334523103	-0.437728632	-0.996644069	-1.419177527	-0.988149164	-0.671576368	0.4269309155	-0.340829438
11														
12	0.1827758064	-0.711270667	0.0645867856	0.2983359265	0.1712644644	-0.345644403	0.5303979379	1.1563058227	-1.131202002	-0.650795055	-0.012019925	-2.1732011682	-0.856558826	-0.158294855
13														
14	-1.488178178	0.2372209677	-0.930755038	0.9931255591	1.6912744284	0.0784920587	1.3629822506	-1.326159866	0.2461492801	0.4445044919	-0.8261831311	-1.540295811	0.1182761964	1.2260798285
15														
16	-2.036741048	-0.002105221	0.2744828738	0.2410225041	0.9889767955	-0.569787173	0.7602597913	-0.589545643	0.3059953013	0.3576908781	0.1828781621	1.7725408671	1.103970499	-0.4113568017
17														
18	1.4773836881	-0.296554184	1.5154911477	-1.730576921	2.803917331	0.4721851478	0.865252622	-0.260985754	-0.070494736	0.9916269839	0.3395495899	0.4636974155	-1.103382636	0.4138980227
19														
20	-1.468736096	-0.083625447	2.0417857937	-0.007696229	0.3957761587	-0.301046156	0.4873504846	-0.218460241	1.7766897255	0.0176370257	2.5314053684	-2.222165887	-0.801740275	0.1917749709
21														
22	-1.681580378	-0.51509047	-0.264592543	0.4455685172	-1.129807941	-0.036632201	-1.097493406	-0.338845238	0.4208150683	-1.241088595	0.0253358161	0.965347029	-0.913625802	-0.229491998
23														
24	0.1619743604	0.5620424384	0.9688980108	2.115576315	0.2991857929	-0.405874774	1.6773343197	0.7221909303	0.7211245658	-1.744402919	0.3311288658	0.5966322361	0.0351372696	0.1830554802
25														
26	-0.3957114745	1.1497833665	-0.584491534	0.1361121799	-1.640638394	-0.111104031	1.279185065	-0.202671285	0.7456430098	0.5159115318	0.7442775618	0.3803240696	1.5143013803	0.4745007834
27														
28	0.0387166724	-0.001416976	0.3460193433	-2.244770075	0.9037821034	-0.360880724	-0.94534609	-1.715073473	1.6233718984	0.7287868281	0.3486613372	-0.982024612	-0.5078358116	-0.1087711322
29														
30	1.0033771423	0.7859569322	-0.423997664	0.1095731108	-0.218971954	-0.435593926	-0.862413686	1.3922128945	1.0347715195	-0.638762032	-1.127701529	-0.70532704	0.2813181439	0.4581019207
31														
32	0.3261368793	0.7628027469	-0.464221255	1.1524295902	-0.058002705	0.1021436631	2.3221692058	-1.33194504	0.7958635389	2.2150290735	0.257114086	-1.241677836	1.716149775	1.6597731148
33														
34	0.1711146361	0.3500225291	0.5168123054	2.0412777272	0.1769730663	-0.204966166	0.2068002094	-1.140075285	0.4220651192	-1.367048317	0.0322674377	-1.394209607	-0.730295886	0.6915960568
35														
36	-0.0933110163	0.1118967817	0.9033396647	-0.504689405	0.059384883	-0.541378811	0.3356334901	1.8316770292	-0.224768755	1.0772798162	-0.313975453	0.291684332	0.325880003	-0.190176409

Figura 3.7: Datos guardados en el archivo CVS para la prueba aleatoria.

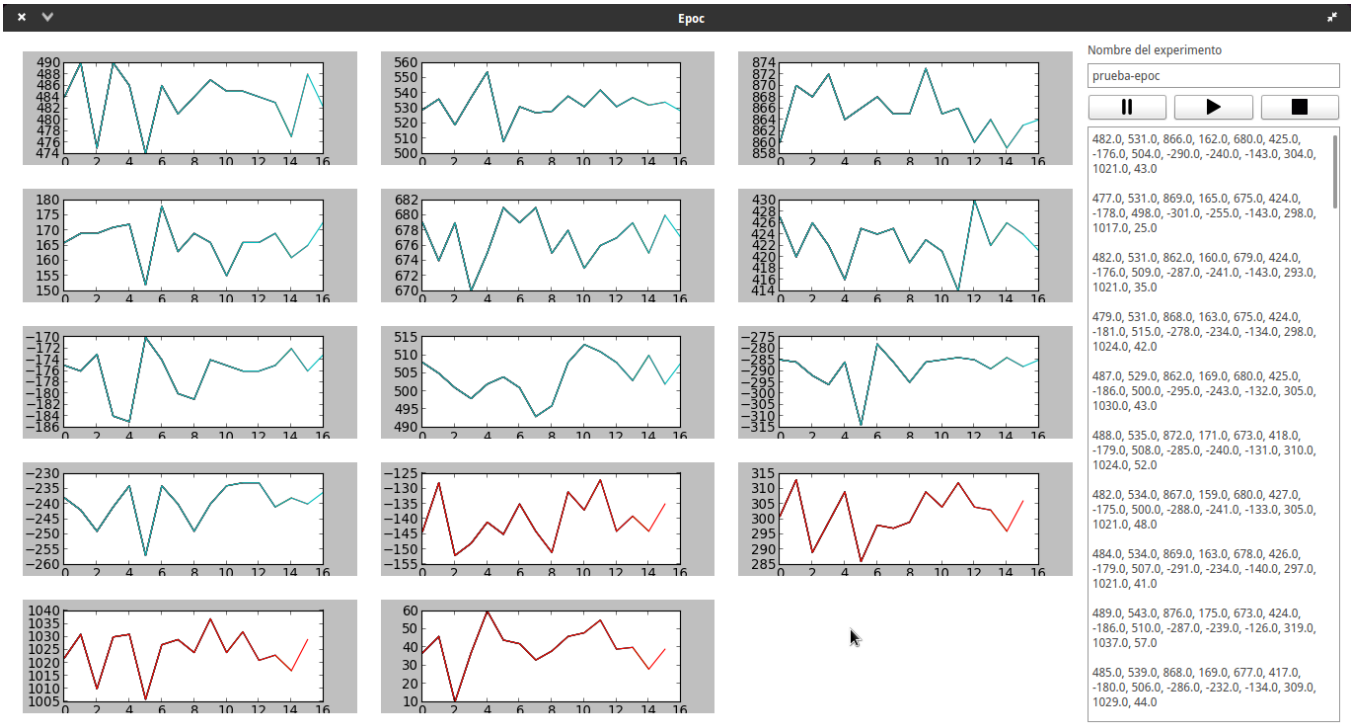

Figura 3.8: Interfaz graficando las señales recibidas del Emokit.

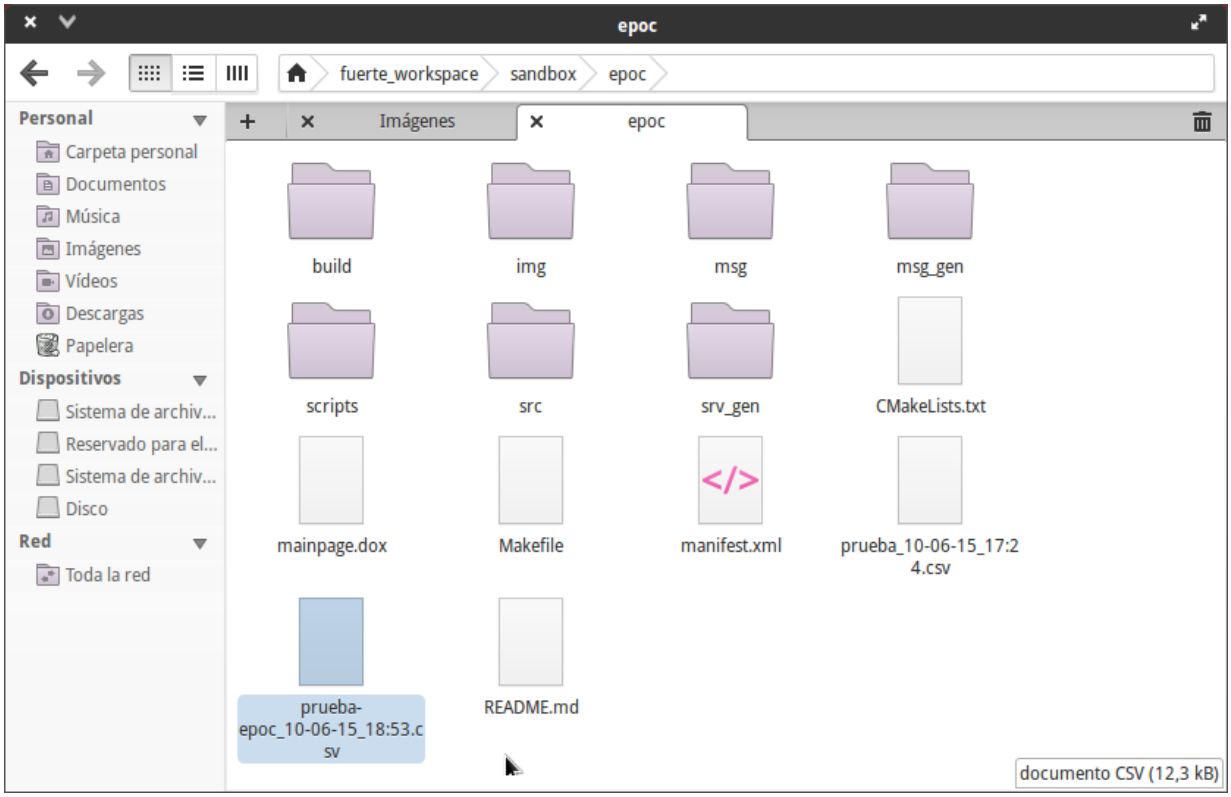

Figura 3.9: Archivo CVS creado para la prueba con el Emokit.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	F3	FC5	AF3	F7	T7	P7	O1	O2	P8	T8	F8	AF4	FC6	F4
2	482	531	866	162	680	425	-176	504	-290	-240	-143	304	1021	43
3														
4	477	531	869	165	675	424	-178	498	-301	-255	-143	298	1017	25
5														
6	482	531	862	160	679	424	-176	509	-287	-241	-143	293	1021	35
7														
8	479	531	868	163	675	424	-181	515	-278	-234	-134	298	1024	42
9														
10	487	529	862	169	680	425	-186	500	-295	-243	-132	305	1030	43
11														
12	488	535	872	171	673	418	-179	508	-285	-240	-131	310	1024	52
13														
14	482	534	867	159	680	427	-175	500	-288	-241	-133	305	1021	48
15														
16	484	534	869	163	678	426	-179	507	-291	-234	-140	297	1021	41
17														
18	489	543	876	175	673	424	-186	510	-287	-239	-126	319	1037	57
19														
20	485	539	868	169	677	417	-180	506	-286	-232	-134	309	1029	44
21														
22	487	536	873	165	676	421	-181	503	-283	-238	-133	305	1028	46
23														
24	479	536	869	162	678	421	-174	510	-285	-245	-135	305	1027	42
25														
26	482	527	867	166	671	424	-169	512	-283	-232	-139	299	1031	44
27														
28	485	535	865	170	677	427	-175	510	-274	-236	-135	301	1024	43

Figura 3.10: Datos guardados en el archivo CVS para la prueba con el Emokit.

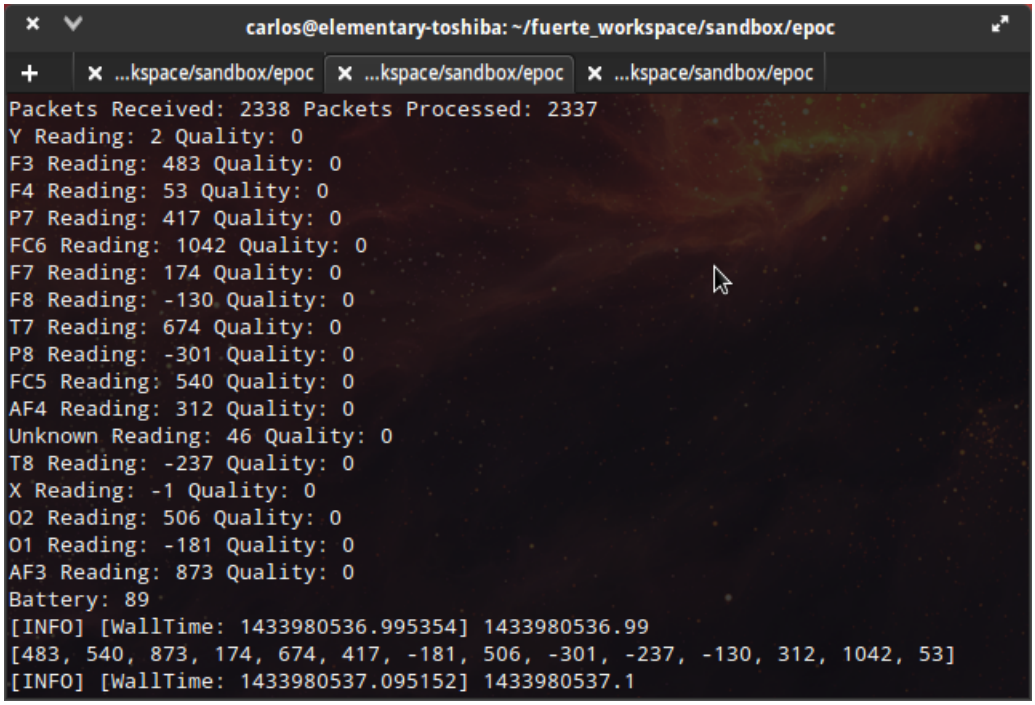

Figura 3.11: Información mostrada a la primera ejecución el Emokit.

Figura 3.12: Impresión de los datos leídos por el Emokit.

Capítulo 4

Conclusión y trabajo futuro

El Emotiv Epoc es una interfaz cerebro-máquina comercial de bajo costos usada para entretenimiento e investigación que cuenta con su propio kit de desarrollo privativo. Debido a las limitaciones del kit de desarrollo privativo surgen necesidades en el ámbito de la ciencia de tener software libre y adaptable a las necesidades específicas de la investigación. El Emokit es un software de código libre creado para ser una alternativa al momento de leer las señales emitidas a través del Epoc. La nueva interfaz fue creada para funcionar junto con el Emokit para ofrecer nuevas características. La nueva interfaz es capaz de graficar, mostrar y almacenar las señales obtenidas del Epoc mediante el Emokit además de ofrecer botones para que el usuario tenga control sobre que muestra la interfaz. La interfaz desarrollada en este trabajo concluye con las siguientes características y funciones:

Características

- Libre y de código abierto.
- Multiplataforma: Al estar construido en Python y PyQt.
- Multipropósito: Aunque fue creado para funcionar con el Emokit el software puede ser usado para cualquier mensaje en ROS que sea una lista de 14 datos flotantes.
- Interfaz Natural: El diseño de la interfaz se adapta al Sistema Operativo donde se esté ejecutando.

Funciones

- Graficación de las señales: Una gráfica para cada señal.
- Visualización de las señales recibidas: El cuadro de log permite ver las señales que se estén recibiendo en una sesión.
- Botones de control de usuario: Con tres botones de “Reproducir”, “Pausar” y “Detener” que dan control al usuario sobre la visualización de los datos.

- Almacenamiento en disco de los datos: Al presionar el botón “Detener” se guardará automáticamente en la carpeta del proyecto un archivo CVS construido con el nombre ingresado en el cuadro de texto correspondiente mas la fecha y hora del sistema.
- Dar un nombre a un set de datos: Ingresando un nombre en el cuadro de texto para esto las señales que se registren desde que se presiones “Reproducir” hasta que se presione “Detener” se guardaran como se menciona en el punto anterior.

4.1. Trabajo Futuro

Para un panorama más real en el manejo de los datos será necesario resolver el problema de velocidad de la graficación. Esta velocidad es afectada por el número de gráficas que se encuentren en la interfaz y por la librería matplotlib en sí. Para solucionarlo será necesario buscar alternativas a esta librería que permitan una velocidad de graficación más rápida. Esto afectará directamente a la cantidad de señales leídas en un determinado tiempo. Se puede también implementar gráficas interactivas donde, por ejemplo, al colocar el cursor en un punto te indique su valor.
Una gran mejora a la interfaz puede ser la incorporación de un módulo de entrenamiento donde por medio de un usuario el software pueda identificar a la persona e ir registrando y aprendiendo en cada sesión que esta realice. Esto combinado con un módulo de interpretación de datos puede ayudar a conocer la intención o pensamiento de la persona.
A partir de la implementación de los módulos anteriores se puede crear uno mas complejo que interprete los movimiento que la persona piense y los reproduzca en un ambiente virtual, robots, brazos mecánicos, drones, etc.

Índice de figuras

Índice de cuadros

Bibliografía

[1] C. Brocious and K. Machulis, "Emokit." http://www.openyou.org, May 2015.

[2] K. Roebuck, *Brain-Computer Interface: High-impact Emerging Technology - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Publishing, 2012.

[3] M. Montalvo, "Estado del arte: Interfaces cerebro computadora."

[4] G. M. L. Anupama. H. S, Cauvery. N. K, "Brain computer interface and its types - a study," 2012.

[5] T. N. Lal, T. Hinterberger, G. Widman, M. Schröder, J. Hill, W. Rosenstiel, C. Elger, B. Schölkopf, and N. Birbaumer, "Methods towards invasive human brian computer interfaces," 2005.

[6] V. Autores, "Biofeedback federation of europe." https://biofeedbackfederationofeurope.blogspot.mx/2014/04/introduction-to-slow-cortical.html, May 2015.

[7] N. Birbaumer, "Breaking the silence: brain–computer interfaces (bci) for communication and motor control," *Psychophysiology*, vol. 43, no. 6, pp. 517–532, 2006.

[8] M. Á. L. Gordo, "Interfaz bci de altas prestaciones basada en la detección y procesamiento de la actividad cerebral (bci-depracap)," 2009.

[9] J. D. Asimbaya Molina and J. A. Suasnavas Tipán, "Diseño e implementación de un prototipo brain computer interface (bci), para la manipulación de una pinza robótica utilizando comunicación bluetooth.," 2014.

[10] J. R. de la O Chávez, "Interfaz cerebro – computadora para el control de un cursor basado en ondas cerebrales," pp. 16–19.

[11] J. D. Bayliss and D. H. Ballard, "A virtual reality testbed for brain-computer interface research," *Rehabilitation Engineering, IEEE Transactions on*, vol. 8, no. 2, pp. 188–190, 2000.

[12] K. LaFleur, K. Cassady, A. Doud, K. Shades, E. Rogin, and B. He, "Quadcopter control in three-dimensional space using a noninvasive motor imagery-based brain-computer interface," *Journal of Neural Engineering*, 2013.

[13] V. Autores, "Emotiv systems." `http://en.wikipedia.org/wiki/Emotiv_Systems`, May 2015.

[14] "Emotiv epoc & testbench specifications." `https://emotiv.com/product-specs/Emotiv%20EPOC%20Specifications%202014.pdf`, June 2015.

[15] K. Stytsenko, E. Jablonskis, and C. Prahm, "Evaluation of consumer eeg device emotiv epoc," in *MEi: CogSci Conference 2011, Ljubljana*, 2011.

[16] V. Autores, "G-tec biosignal amplifier g.bsamp." `http://www.gtec.at/Products/Hardware-and-Accessories/g.BSamp-Specs-Features`, May 2015.

[17] "Emotiv sdk." `http://innovatec.co.jp/content/etc/ResearchEditionSDK.pdf`, June 2015.

[18] F. J. Muñoz Sánchez, "Construcción de una nueva interfaz cerebro-computadora a partir de una de bajo coste (emotiv epoc)," 2014.

[19] I. Senior Design, "Epoc-alypse mind controlled car," 2012.

[20] A. Raza, "Ssvep based eeg interface for google street view navigation," 2012.

[21] J. Castillo, B. Longo, A. Floriano, E. Caicedo, and T. Bastos, "Optimización de una interfaz cerebro computador basada en imaginación motora usando emotiv epoc,"

[22] J. S. Ierache, G. Pereira, J. Iribarren, and F. Nervo, "Estado emocional centrado en estímulos, aplicando interfase cerebro-maquina," in *XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014)*, 2014.

[23] U. Loskit, "Signal quality and data visualizer for emotiv epoc," 2014.

[24] V. Autores, "Ros.org." `http://wiki.ros.org/es`, May 2015.

[25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.

[26] M. Kranz, L. Roalter, and F. Michahelles, "Things that twitter: social networks and the internet of things," in *What can the Internet of Things do for the Citizen (CIoT) Workshop at The Eighth International Conference on Pervasive Computing (Pervasive 2010)*, pp. 1–10, 2010.

[27] M. F. Sanner *et al.*, "Python: a programming language for software integration and development," *J Mol Graph Model*, vol. 17, no. 1, pp. 57–61, 1999.

Capítulo 5

Apéndice

5.1. Códigos

5.1.1. interfaz.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import roslib; roslib.load_manifest('epoc')
from epoc.msg import Frecuencias
from GUI import *
import sys
from time import strftime
from time import sleep
import datetime

class GUIForm(QtGui.QWidget):

    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self,parent)

        # Inicializacion de la interfaz
        self.ui = Ui_EpocGUI()
        self.ui.setupUi(self)

        # Bandera para permite/impedir graficacion
        self.grafica = False

        # Lista de frecuencias
        self.frecuencias = [[],[],[],[],[],[],[],[],[],[],[],[],[]]

        # Comienza a escuchar los mensajes en ROS
        self.listener()
        self.tiempo_i = datetime.datetime.now()
        self.tiempo_f = datetime.datetime.now()
```

```
32          self.ocurrencias = 0
33
34          # Definicion de los botones
35          QtCore.QObject.connect(self.ui.botonEjecutar, QtCore.SIGNAL('clicked()
    '), self.habilitaGraficar)
36          QtCore.QObject.connect(self.ui.botonDetener, QtCore.SIGNAL('clicked()'
    ), self.detenerGraficar)
37          QtCore.QObject.connect(self.ui.botonPausar, QtCore.SIGNAL('clicked()')
    , self.pausaGraficar)
38
39      def habilitaGraficar(self):
40          """ Habilita la graficacion por medio de la bandera """
41
42          self.grafica = True
43          self.tiempo_i = datetime.datetime.now()
44          self.ocurrencias = 0
45
46      def graficar(self, frecuencias):
47          """ Funcion que manda los datos a graficar """
48
49          # Anade los puntos a la grafica
50          self.ui.widget.canvas.ax.plot(frecuencias[0])
51          self.ui.widget_2.canvas.ax.plot(frecuencias[1])
52          self.ui.widget_3.canvas.ax.plot(frecuencias[2])
53          self.ui.widget_4.canvas.ax.plot(frecuencias[3])
54          self.ui.widget_5.canvas.ax.plot(frecuencias[4])
55          self.ui.widget_6.canvas.ax.plot(frecuencias[5])
56          self.ui.widget_7.canvas.ax.plot(frecuencias[6])
57          self.ui.widget_8.canvas.ax.plot(frecuencias[7])
58          self.ui.widget_9.canvas.ax.plot(frecuencias[8])
59          self.ui.widget_10.canvas.ax.plot(frecuencias[9])
60          self.ui.widget_11.canvas.ax.plot(frecuencias[10])
61          self.ui.widget_12.canvas.ax.plot(frecuencias[11])
62          self.ui.widget_13.canvas.ax.plot(frecuencias[12])
63          self.ui.widget_14.canvas.ax.plot(frecuencias[13])
64
65          # Realiza el dibujado en la grafica
66          self.ui.widget.canvas.draw()
67          self.ui.widget_2.canvas.draw()
68          self.ui.widget_3.canvas.draw()
69          self.ui.widget_4.canvas.draw()
70          self.ui.widget_5.canvas.draw()
71          self.ui.widget_6.canvas.draw()
72          self.ui.widget_7.canvas.draw()
73          self.ui.widget_8.canvas.draw()
74          self.ui.widget_9.canvas.draw()
75          self.ui.widget_10.canvas.draw()
76          self.ui.widget_11.canvas.draw()
77          self.ui.widget_12.canvas.draw()
78          self.ui.widget_13.canvas.draw()
79          self.ui.widget_14.canvas.draw()
```

```
80
81      def detenerGraficar(self):
82          """ Detiene la graficacion y genera el log """
83
84          self.grafica = False
85          self.tiempo_f = datetime.datetime.now()
86          print 'tiempo_i: %s' % self.tiempo_i
87          print 'tiempo_f: %s' % self.tiempo_f
88          print 'diferencia: %s' % (self.tiempo_f - self.tiempo_i)
89          print 'Ocurrencias: ' + str(self.ocurrencias)
90          self.generarLog()
91
92      def pausaGraficar(self):
93          """ Pausa la graficacion """
94          self.grafica = False
95
96      def generarLog(self):
97          sleep(.1)
98
99          # Extraccion del nombre del cuadro de texto
100         nombre = self.ui.textNombrePrueba.toPlainText()
101
102         # Valida que se haya ingresado un nombre de prueba
103         if nombre:
104             nombre = nombre + '_' + strftime("%d-%m-%y_%H:%M") + '.csv'
105             file = open(nombre, 'w')
106             file.write('F3, FC5, AF3, F7, T7, P7, O1, O2, P8, T8, F8, AF4, FC6
    , F4\n')
107             file.write(self.ui.textBrowser.toPlainText())
108             self.ui.textBrowser.setPlainText("")
109         else:
110             msgBox = QtGui.QMessageBox(QtGui.QMessageBox.Warning,
111                 "QMessageBox.warning()", "Espeficifica un nombre y presiona
    Detener nuevamente",
112                 QtGui.QMessageBox.NoButton, self)
113             msgBox.exec_()
114
115     def callback(self, data):
116         """ Recibe el mensaje transmitido """
117
118         if self.grafica:
119             data = list(data.datos)
120             self.ocurrencias += 1
121
122             # La lista frecuencias va encolando las senales en cada iteracion
123             # data contiene las 14 senales de la iteracion actual
124             self.frecuencias[0].append(data[0])
125             self.frecuencias[1].append(data[1])
126             self.frecuencias[2].append(data[2])
127             self.frecuencias[3].append(data[3])
128             self.frecuencias[4].append(data[4])
```

```
129             self.frecuencias[5].append(data[5])
130             self.frecuencias[6].append(data[6])
131             self.frecuencias[7].append(data[7])
132             self.frecuencias[8].append(data[8])
133             self.frecuencias[9].append(data[9])
134             self.frecuencias[10].append(data[10])
135             self.frecuencias[11].append(data[11])
136             self.frecuencias[12].append(data[12])
137             self.frecuencias[13].append(data[13])
138 
139             # Arma el texto para mostrarlo en el cuadro de log
140             log = str(str(data[0]) + ', ' + str(data[1]) + ', ' + str(data[2])
    +
141                       ', ' + str(data[3]) + ', ' + str(data[4]) + ', ' + str(
    data[5]) +
142                       ', ' + str(data[6]) + ', ' + str(data[7]) + ', ' + str(
    data[8]) +
143                       ', ' + str(data[9]) + ', ' + str(data[10]) + ', ' + str(
    data[11]) +
144                       ', ' + str(data[12]) + ', ' + str(data[13]))
145 
146             self.ui.textBrowser.appendPlainText(log)
147             self.ui.textBrowser.appendPlainText("")
148 
149             # Resetea la lista de frecuencias y limpia las graficas cuando la
    lista llega a los 30 elementos
150             if len(self.frecuencias[0]) == 30:
151                 self.frecuencias = [[],[],[],[],[],[],[],[],[],[],[],[],[]]
152                 self.ui.widget.canvas.ax.clear()
153                 self.ui.widget_2.canvas.ax.clear()
154                 self.ui.widget_3.canvas.ax.clear()
155                 self.ui.widget_4.canvas.ax.clear()
156                 self.ui.widget_5.canvas.ax.clear()
157                 self.ui.widget_6.canvas.ax.clear()
158                 self.ui.widget_7.canvas.ax.clear()
159                 self.ui.widget_8.canvas.ax.clear()
160                 self.ui.widget_9.canvas.ax.clear()
161                 self.ui.widget_10.canvas.ax.clear()
162                 self.ui.widget_11.canvas.ax.clear()
163                 self.ui.widget_12.canvas.ax.clear()
164                 self.ui.widget_13.canvas.ax.clear()
165                 self.ui.widget_14.canvas.ax.clear()
166 
167             self.graficar(self.frecuencias)
168 
169     def listener(self):
170         """ Inicia la comunicacion ROS """
171 
172         rospy.init_node('listener', anonymous=True, disable_signals=False)
173 
```

```
174         # Se suscribe a un mensaje de ROS, el primer parametro identifica al
    mensaje,
175         # el segundo es el nombre del mensaje definido en la carpeta msg del
    proyecto,
176         # el tercer parametro es la llamada al metodo callback que se
    encargara de estar leyendo siempre los mensajes
177         rospy.Subscriber("mensaje", Frecuencias, self.callback)
178
179
180 if __name__ == '__main__':
181
182     app = QtGui.QApplication(sys.argv)
183     myapp = GUIForm()
184     myapp.show()
185     sys.exit(app.exec_())
```

Listing 5.1: interfaz.py

5.1.2. GUI.py

```
# -*- coding: utf-8 -*-

from PyQt4 import QtCore, QtGui
from PyQt4.QtGui import QHBoxLayout, QVBoxLayout, QGridLayout
from matplotlibwidgetFile import matplotlibWidget

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    _fromUtf8 = lambda s: s

class Ui_EpocGUI(object):
    def setupUi(self, EpocGUI):

        # Definicion de las layouts
        EpocGUI.main_layout = QHBoxLayout()
        self.layout_izq = QVBoxLayout()
        self.layout_der = QVBoxLayout()
        self.grid = QtGui.QGridLayout()
        self.grid_botones = QtGui.QGridLayout()

        EpocGUI.setObjectName(_fromUtf8("EpocGUI"))
        EpocGUI.resize(1101, 897) # Resolucion por defecto

        # Definicion de las graficas
        self.centralwidget = QtGui.QWidget(EpocGUI)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))

        self.widget = matplotlibWidget(self.centralwidget)
        self.widget.setObjectName(_fromUtf8("widget"))

        self.widget_2 = matplotlibWidget(self.centralwidget)
        self.widget_2.setObjectName(_fromUtf8("widget_2"))

        self.widget_3 = matplotlibWidget(self.centralwidget)
        self.widget_3.setObjectName(_fromUtf8("widget_3"))

        self.widget_4 = matplotlibWidget(self.centralwidget)
        self.widget_4.setObjectName(_fromUtf8("widget_4"))

        self.widget_5 = matplotlibWidget(self.centralwidget)
        self.widget_5.setObjectName(_fromUtf8("widget_5"))
        self.widget_6 = matplotlibWidget(self.centralwidget)
        self.widget_6.setObjectName(_fromUtf8("widget_6"))

        self.widget_7 = matplotlibWidget(self.centralwidget)
        self.widget_7.setObjectName(_fromUtf8("widget_7"))

        self.widget_8 = matplotlibWidget(self.centralwidget)
```

```
50          self.widget_8.setObjectName(_fromUtf8("widget_8"))
51
52          self.widget_9 = matplotlibWidget(self.centralwidget)
53          self.widget_9.setObjectName(_fromUtf8("widget_9"))
54
55          self.widget_10 = matplotlibWidget(self.centralwidget)
56          self.widget_10.setObjectName(_fromUtf8("widget_10"))
57
58          self.widget_11 = matplotlibWidget(self.centralwidget)
59          self.widget_11.setObjectName(_fromUtf8("widget_11"))
60
61          self.widget_12 = matplotlibWidget(self.centralwidget)
62          self.widget_12.setObjectName(_fromUtf8("widget_12"))
63
64          self.widget_13 = matplotlibWidget(self.centralwidget)
65          self.widget_13.setObjectName(_fromUtf8("widget_13"))
66
67          self.widget_14 = matplotlibWidget(self.centralwidget)
68          self.widget_14.setObjectName(_fromUtf8("widget_14"))
69
70          # Definicion de los botones
71          self.botonPausar = QtGui.QPushButton(self.centralwidget)
72          self.botonPausar.setText(_fromUtf8(""))
73          icon = QtGui.QIcon()
74          icon.addPixmap(QtGui.QPixmap(_fromUtf8("img/pause.png")), QtGui.QIcon.
    Normal, QtGui.QIcon.Off)
75          self.botonPausar.setIcon(icon)
76          self.botonPausar.setObjectName(_fromUtf8("botonPausar"))
77
78          self.botonDetener = QtGui.QPushButton(self.centralwidget)
79          self.botonDetener.setText(_fromUtf8(""))
80          icon1 = QtGui.QIcon()
81          icon1.addPixmap(QtGui.QPixmap(_fromUtf8("img/stop.png")), QtGui.QIcon.
    Normal, QtGui.QIcon.Off)
82          self.botonDetener.setIcon(icon1)
83          self.botonDetener.setObjectName(_fromUtf8("botonDetener"))
84
85          self.botonEjecutar = QtGui.QPushButton(self.centralwidget)
86          self.botonEjecutar.setText(_fromUtf8(""))
87          icon2 = QtGui.QIcon()
88          icon2.addPixmap(QtGui.QPixmap(_fromUtf8("img/play.png")), QtGui.QIcon.
    Normal, QtGui.QIcon.Off)
89          self.botonEjecutar.setIcon(icon2)
90          self.botonEjecutar.setObjectName(_fromUtf8("botonEjecutar"))
91
92          # Definicion del cuadro de log
93          self.textBrowser = QtGui.QPlainTextEdit(self.centralwidget)
94          self.textBrowser.setObjectName(_fromUtf8("textBrowser"))
95          self.textBrowser.setReadOnly(True)
96          self.textBrowser.setSizePolicy(QtGui.QSizePolicy.Minimum, QtGui.
    QSizePolicy.Expanding)
```

```
97
98          # Definicion de cuadro y etiqueta de nombre
99          self.textNombrePrueba = QtGui.QTextEdit(self.centralwidget)
100         self.textNombrePrueba.setObjectName(_fromUtf8("textEdit_2"))
101         self.textNombrePrueba.setMaximumHeight(25)
102
103         self.label = QtGui.QLabel(self.centralwidget)
104         self.label.setObjectName(_fromUtf8("label"))
105
106         self.menubar = QtGui.QMenuBar(EpocGUI)
107         #self.menubar.setGeometry(QtCore.QRect(0, 0, 1101, 23))
108         self.menubar.setObjectName(_fromUtf8("menubar"))
109
110         # Agrega las graficas a un gridLayout
111         #EpocGUI.main_layout.addWidget(self.centralwidget)
112         self.grid.addWidget(self.widget,0,0)
113         self.grid.addWidget(self.widget_2,0,1)
114         self.grid.addWidget(self.widget_3,0,2)
115         self.grid.addWidget(self.widget_4,1,0)
116         self.grid.addWidget(self.widget_5,1,1)
117         self.grid.addWidget(self.widget_6,1,2)
118         self.grid.addWidget(self.widget_7,2,0)
119         self.grid.addWidget(self.widget_8,2,1)
120         self.grid.addWidget(self.widget_9,2,2)
121         self.grid.addWidget(self.widget_10,3,0)
122         self.grid.addWidget(self.widget_11,3,1)
123         self.grid.addWidget(self.widget_12,3,2)
124         self.grid.addWidget(self.widget_13,4,0)
125         self.grid.addWidget(self.widget_14,4,1)
126
127         # Agrega las graficas a la parte izquierda de la pantalla
128         self.layout_izq.addLayout(self.grid)
129
130         # Agrega elementos de la parte derecha de la pantalla
131         self.layout_der.addWidget(self.label)
132         self.layout_der.addWidget(self.textNombrePrueba)
133
134         self.grid_botones.addWidget(self.botonPausar,0,0)
135         self.grid_botones.addWidget(self.botonEjecutar,0,1)
136         self.grid_botones.addWidget(self.botonDetener,0,2)
137         self.layout_der.addLayout(self.grid_botones)
138
139         self.layout_der.addWidget(self.textBrowser)
140
141         # Agrega los layouts izquierdo y derecho al principal
142         EpocGUI.main_layout.addLayout(self.layout_izq)
143         EpocGUI.main_layout.addLayout(self.layout_der)
144         EpocGUI.setLayout(EpocGUI.main_layout)
145
146         self.retranslateUi(EpocGUI)
147         QtCore.QMetaObject.connectSlotsByName(EpocGUI)
```

```
148
149     def retranslateUi(self, EpocGUI):
150         EpocGUI.setWindowTitle(QtGui.QApplication.translate("EpocGUI", "Epoc",
        None, QtGui.QApplication.UnicodeUTF8))
151         self.label.setText(QtGui.QApplication.translate("EpocGUI", "Nombre del
        experimento", None, QtGui.QApplication.UnicodeUTF8))
```

Listing 5.2: GUI.py

5.1.3. matplotlibwidgetFile.py

```
from PyQt4 import QtGui
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as
    FigureCanvas
import matplotlib.pyplot as plt

from matplotlib.figure import Figure

plt.show(block=False)

class MplCanvas(FigureCanvas):

    def __init__(self):
        #self.fig = Figure()
        #self.ax = self.fig.add_subplot(111)
        self.fig, self.ax = plt.subplots()

        FigureCanvas.__init__(self, self.fig)
        FigureCanvas.setSizePolicy(self, QtGui.QSizePolicy.Expanding,QtGui.
    QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)


class matplotlibWidget(QtGui.QWidget):

    def __init__(self, parent = None):
        QtGui.QWidget.__init__(self, parent)
        self.canvas = MplCanvas()
        self.vbl = QtGui.QVBoxLayout()
        self.vbl.addWidget(self.canvas)
        self.setLayout(self.vbl)
```

Listing 5.3: matplotlibwidgetFile.py

5.1.4. emokit.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Librerias de ROS
import rospy
import numpy as np
import roslib; roslib.load_manifest('epoc')
from epoc.msg import Frecuencias

# Librerias de Emokit
from emotiv import Emotiv
import platform
if platform.system() == "Windows":
    import socket  # Needed to prevent gevent crashing on Windows. (surfly / gevent issue #459)
import gevent

def obtenerDatos(datos):
    """ Obtiene los datos leidos por el Emotiv Epoc y los almacena y retorna en la lista datos[] """

    # Obtiene el paquete del Epoc
    packet = headset.dequeue()

    # Obtiene los datos que vienen como parte del paquete obtenido
    datos.append(packet.sensors['F3']['value'])
    datos.append(packet.sensors['FC5']['value'])
    datos.append(packet.sensors['AF3']['value'])
    datos.append(packet.sensors['F7']['value'])
    datos.append(packet.sensors['T7']['value'])
    datos.append(packet.sensors['P7']['value'])
    datos.append(packet.sensors['O1']['value'])
    datos.append(packet.sensors['O2']['value'])
    datos.append(packet.sensors['P8']['value'])
    datos.append(packet.sensors['T8']['value'])
    datos.append(packet.sensors['F8']['value'])
    datos.append(packet.sensors['AF4']['value'])
    datos.append(packet.sensors['FC6']['value'])
    datos.append(packet.sensors['F4']['value'])
    print datos
    gevent.sleep(0)

def talker():
    """ Funcion para publicar los mensajes en ROS """

    # Configura la funcion para que publique los mensajes.
    # El primer parametro es el nombre para identificar el mensaje,
    # el segundo es el nombre del mensaje definido en la carpeta msg del proyecto
```

```
47      pub = rospy.Publisher('mensaje', Frecuencias)
48      rospy.init_node('talker', anonymous=True, disable_signals=False) #
    Inicializa el nodo
49      rate = rospy.Rate(10) # 10hz # Define la velocidad de publicacion en ROS
50
51      # Publicara los mensajes mientras que no se apague la comunicacion
52      while not rospy.is_shutdown():
53          datos = []
54          hello_str = "%s" %rospy.get_time()
55          rospy.loginfo(hello_str)
56
57          obtenerDatos(datos) # Obtiene los datos del Epoc
58
59          pub.publish(datos) # Publica el mensaje en ROS, el parametro debe ser
    del mismo tipo definido en el archivo de mensaje
60          rate.sleep()
61
62
63  if __name__ == '__main__':
64
65      # Inicia la comunicacion con el Epoc por medio del Emokit
66      headset = Emotiv()
67      gevent.spawn(headset.setup)
68      gevent.sleep(0)
69
70      try:
71          talker()
72      except rospy.ROSInterruptException:
73          pass
74      except KeyboardInterrupt:
75          headset.close()
76      finally:
77          headset.close()
```

Listing 5.4: emokit.py